Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Análisis de prestaciones del protocolo HTTP2

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autores: Javier Marín Pérez

Tutor: Oliver Gil, José Salvador

2017/2018

Resumen

El trabajo realizado muestra un análisis realizado sobre el Protocolo de Transferencia de Hipertexto, versión 2, conocido como HTTP/2, que es utilizado en Internet y pretende sustituir a su versión antigua, HTTP/1.1, con el cual es compatible.

Para realizar el análisis, primero se evalúa la complejidad de la transición a HTTP/2, viendo la evolución que se debe implementar para que el servidor trabaje con este protocolo. A continuación, se exponen los cambios que hay en este protocolo. Posteriormente, se cuantifica el tiempo que se tarda en obtener una página mediante HTTP/1.1 y HTTP/2, y se calcula la mejora de rendimiento que se obtiene al implementar este nuevo protocolo. Para finalizar, se expone brevemente el futuro de HTTP/2.

Palabras clave: HTTP/2, protocolo, análisis, rendimiento.

Abstract

The research displayed shows the analysis of Hypertext Transfer Protocol, version 2, known as HTTP/2, which is currently being used for the Internet, and its main objective is to replace its old version, HTTP/1.1, which it is compatible.

To perform this analysis, first, the HTTP/2 transition is evaluated, looking at the changes that must be done to the server, to work with this protocol. Then, the changes that HTTP/2 implements are displayed. Next, the time it takes to obtain a page using HTTP/1.1 and HTTP/2 is saved and used to obtain the performance improvement of this protocol. Finally, the future of HTTP/2 is briefly exposed.

Keywords: HTTP/2, protocol, analysis, performance.

Tabla de contenidos

Tabla de figuras

Tabla de tablas

1. Introducción

La velocidad en los sistemas de telecomunicación es un aspecto a tener en cuenta a la hora de trabajar en informática. Evaluar y calcular la eficiencia de un nuevo protocolo es clave para que sean más rápidas las comunicaciones.

En 2015, se desarrolló el *"Request For Comments"* 7540, el cual establecía el protocolo nuevo de comunicación de hipertexto, HTTP/2, que es compatible con la versión de HTTP/1.1.

HTTP/2 logró mejorar hasta un 60% de la velocidad de carga de las páginas web y aporta la ventaja de compresión de cabeceras conocido como HPACK, por lo que otros protocolos que eran más rápidos que HTTP/1.1 se abandonaron.

1.1 Motivación

La motivación de este proyecto reside en la intención de analizar un protocolo de comunicación nuevo, con el objetivo de comprobar si se mejoran las prestaciones que se exponen en un primer momento, observando posibles fallos de implementación en el protocolo, y calculando numéricamente estas mejoras, obteniendo así, el rendimiento real de la nueva versión.

1.2 Objetivos

El objetivo principal de este proyecto es ver las prestaciones que presenta el nuevo protocolo de hipertexto, conocido como HTTP/2 y comparar los tiempos respecto a su predecesor, HTTP/1.1, estudiando los tiempos de las páginas más visitadas del mundo.

Para ello, primero se exponen los pasos para migrar un servidor a HTTP/2, dando soporte a HTTP/1.1 para los usuarios que no disponen de HTTP/2 y a continuación se captura el tiempo de carga de las páginas web y se obtiene la eficiencia de este nuevo protocolo.

1.3 Estructura de la memoria

La memoria se ha estructurado en cinco partes.

En la primera, se define la historia que sufrido HTTP y los cambios necesarios para migrar un servidor a HTTP, dando soporte también a los usuarios que no disponen de HTTP/1.1.

En la segunda, se exponen los cambios que ha sufrido HTTP en la versión 2, mencionando cada cambio que se ha implantado en la nueva versión y lo que se pretende conseguir con estos cambios.

En la tercera, se describen las principales características que se introducen en HTTP/2 para mejorar el rendimiento de HTTP/1.1

En la cuarta, se analiza el rendimiento de HTTP/2, para ello se valorarán diferentes escenarios, entre ellos, se encuentran análisis simulados, análisis sobre una red física y un análisis de una demo de internet.

En la quinta y última, se explica la conclusión que se extrae del análisis previo para saber bajo qué circunstancias es mejor la versión HTTP/2.

2. Transición a HTTP/2

2.1 Evolución de HTTP

El origen del hipertexto, texto que se muestra en un ordenador u otro dispositivo electrónico con referencias a otro texto que se puede acceder inmediatamente, se remonta a 1945, donde el ingeniero Vannevar Bush escribió un ensayo llamado “As We May Think” donde describía una máquina que podría implementar lo que conocemos actualmente como hipertexto, su nombre era Memex.

Según Bush, Memex era un dispositivo que podía encontrar información en grabaciones, libros y comunicaciones almacenadas en su interior, mejorando el proceso de búsqueda. Para ello, asociaría cada elemento con unos enlaces y anotaciones personales, que podrían ser “llamados” en cualquier momento.

Sin embargo, no sería hasta 1963, cuando Ted Nelson desarrollo un modelo que creaba y usaba contenido entrelazado que llamo hipertexto e hipermedia. La pretensión de Nelson fue crear un documento donde la información estuviera entrelazada y nunca fuera borrada, para que estuviera fácilmente disponible para todo el mundo. Para ello, desarrolló el proyecto Xanadu, el cual fue el primer proyecto que trabajaba con hipertexto. Este proyecto comenzó en 1960, pero no fue hasta 1998 cuando una incompleta implementación fue lanzada. Desafortunadamente, nunca se completó, pero estableció las bases para sostener las versiones futuras.

En 1989, Tim Berners-Lee propuso unir Internet y el hipertexto. Para ello construyo el primer navegador y el primer servidor Web al que llamó httpd. El primer servidor se puso en línea en 1991, y esto proporciono la idea de lo que supondría Internet. Su difusión fue rápida y en 1995 había 200 servidores activos.

HTTP 0.9

La primera versión de HTTP fue lanzada en 1991, y era una aplicación cliente-servidor, que se basaba en una única petición con su correspondiente respuesta, para conectarse con el servidor, era necesario conectarse por telnet.

El único método soportado era el GET, y la petición constaba de una única línea en la cual se le especificaba el método y la ruta del documento deseado. La respuesta era un hipertexto y después de la respuesta, se cerraba la conexión con el servidor.

Puesto que no existían cabeceras, no se podía transmitir otro tipo de archivos, tampoco se podían transmitir códigos de error, ni otras urls.

Un ejemplo de cómo sería la conexión en HTTP/0.9 sería:

1. Conectarse por telnet al servidor:

telnet miservidor.com 80

2. Petición del recurso:

GET /pagina.html

3. Obtención de la respuesta:

<HTML>
Mi página
<HTML>

4. Cierre de conexión después de recibir la respuesta.

HTTP 1.0

En 1996, es lanzada la primera modificación del protocolo. Al contrario que su predecesora, esta usa como método de conexión un navegador web. Un cambio importante que se introduce en esta versión es la inclusión de cabeceras como son el tipo de contenido, el código de estado o la versión de HTTP utilizada.

Al contrario que HTTP/0.9, la inclusión del tipo de contenido permitió utilizar archivos que no fueran HTML. Un ejemplo de esto sería la inclusión de fotografías en la página.

Además del método GET, HTTP/1.0 soportaba los métodos POST y HEAD, con esto se consiguió más interacción entre el cliente y el servidor, al permitir interactuar entre ambos con el nuevo método (POST). Otra adición al protocolo fue la introducción de peticiones condicionales. Sin embargo, tras cada respuesta del servidor la conexión se seguía cerrando.

Un ejemplo de cómo sería una petición en HTTP/1.0 con una conexión preestablecida:

```
GET/pagina.html                                          HTTP/1.0
User-Agent: Mozilla Firefox/7.0
```

Un ejemplo de cómo sería la respuesta en HTTP/1.0 sería:

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 12405
Expires: Mon, 7 May 2018 16:04:00 GMT
Last-Modified: Mon, 7 May 2018 12:00:00 GMT
Server: Apache 0.87

<HTML>
    Hola
    <IMG SRC=”/imagen.jpeg”>

<HTML>
```

HTTP/1.1

Puesto que ambos protocolos anteriores, HTTP/0.9 y HTTP/1.0 tenían un gran problema al tener que establecer una nueva conexión de TCP para cada petición, en 1999 se desarrolló HTTP/1.1.

Esta versión del protocolo introdujo numerosas mejoras y optimizaciones como la introducción de la cabecera “Host”, que permite realizar “host” virtual. Más cabeceras fueron incluidas en el protocolo, como la cabecera “upgrade” y la cabecera “range”, la primera sirve para usar un protocolo más actual una vez se desea y la otra se utiliza actualmente en “streaming”. Otra gran optimización fue el empleo de conexiones persistentes y de “pipelining” (Consiste en enviar varias peticiones a la vez). Además, se introdujo el soporte de la utilización de caché y de la compresión.

Los nuevos métodos añadidos, aparte de los ya mencionados anteriormente, son:

“Put”, “Delete”, “Trace” y “Options”.

SPDY

En 2009, 2 trabajadores de Google, Mike Belshe y Roberto Peon propusieron una alternativa a HTTP llamada SPDY. Los objetivos de SPDY eran bastante claros, su principal objetivo era reducir el tiempo de carga de una página al 50%, evitar hacer cambios al contenido de las páginas web y en los servidores y desarrollar el proyecto con la comunidad de código abierto. Este protocolo no fue el primero en intentar reemplazar a HTTP, pero el esfuerzo que requeriría coordinar todo Internet sería muy complicado.

Sin embargo, SPDY cambió todo. Demostró que había un interés en algo más eficiente y una voluntad hacia el cambio. Promovió ciertas implementaciones que luego fueron usadas en HTTP/2 como la multiplexación o la compresión de cabeceras entre otras. Sin embargo, en 2015, Google anunció que tras la ratificación final del estándar HTTP/2, el soporte para SPDY se considera obsoleto.

HTTP/2

Los cambios que se pretenden abordar en esta nueva versión del protocolo son los siguientes:

-Reducir la latencia que perciben los usuarios.

-Terminar con el "Head Of Line Blocking" en HTTP, que se explica en el apartado 3.

-Usar una única conexión para usar paralelismo, mejorando el uso de TCP.

-Seguir con la semántica de HTTP/1.1 incluyendo los métodos, estados, cabeceras…

-Definir como HTTP/2 y HTTP/1.x interactúan entre sí, especialmente en intermediarios.

Se decidió que el punto de partida para desarrollar HTTP/2 fuera SPDY, por lo tanto, muchas mejoras empleadas en este protocolo fueron usadas en SPDY, como ya se comentó anteriormente.

Finalmente, el 14 de mayo de 2015 se publicó el RFC 7540 y HTTP/2 fue oficial.

2.2 Pasos para obtener HTTP/2

Para obtener un servidor en HTTP/2, se necesita utilizar un servidor que sea capaz de comunicarse en HTTP/2 y necesita un certificado de SSL, un ejemplo de cómo poner en marcha un servidor se ve en el Anexo D, en la página 105.

Consideraciones a tener en cuenta

Además de los pasos anteriores para obtener un servidor que funcione con HTTP/2, hay que tener en cuenta antes de usar HTTP/2 en una página web.

El primer paso es el soporte que ofrecen los navegadores web. Actualmente, alrededor del 80% de los navegadores web soportan el protocolo HTTP/2, y además puesto que no implica al usuario en la negociación del protocolo y es transparente a los navegadores que no soportan HTTP/2, simplemente acceden a la página web mediante HTTP/1.1 si no existe soporte para HTTP/2 [11].

La tabla 1 muestra los distintos navegadores web y su soporte parcial o total.

Tabla 1 Soporte de HTTP/2 en los navegadores web

Navegador	Versión Actual	Versión en la que soporta HTTP2	Fecha de la versión
Internet Explorer	11	11 (Obligatorio Windows 10)	17 octubre 2013
Edge	17	12	29 julio 2015
Firefox	59	36 *	24 febrero 2015
Chrome	66	41 *	5 marzo 2015
Safari	11.1	9 (Obligatorio OSX 10.11+)	1 octubre 2015
		11	19 septiembre 2017
Opera	50	28 *	10 marzo 2015
iOS Safari	11.3	9.2	16 septiembre 2015
Opera Mini	32		
Android	62	62	1 febrero 2017
BlackBerry	10		
Opera Mobile	37	37	23 septiembre 2016
Chrome Android	66	66 *	17 abril 2018
Firefox Android	57	57	14 noviembre 2017
IE Mobile	11		
UC for Mobile	11.8		
Samsung Internet	6.2	4 *	19 abril 2016
QQ	1.2	1.2 *	1 enero 2017
Baidu	7.12	7.12	1 abril 2017

Soportado Totalmente (Requiere TLS)	Soportado Parcialmente (TLS + Otro requisito)	No soportan el protocolo

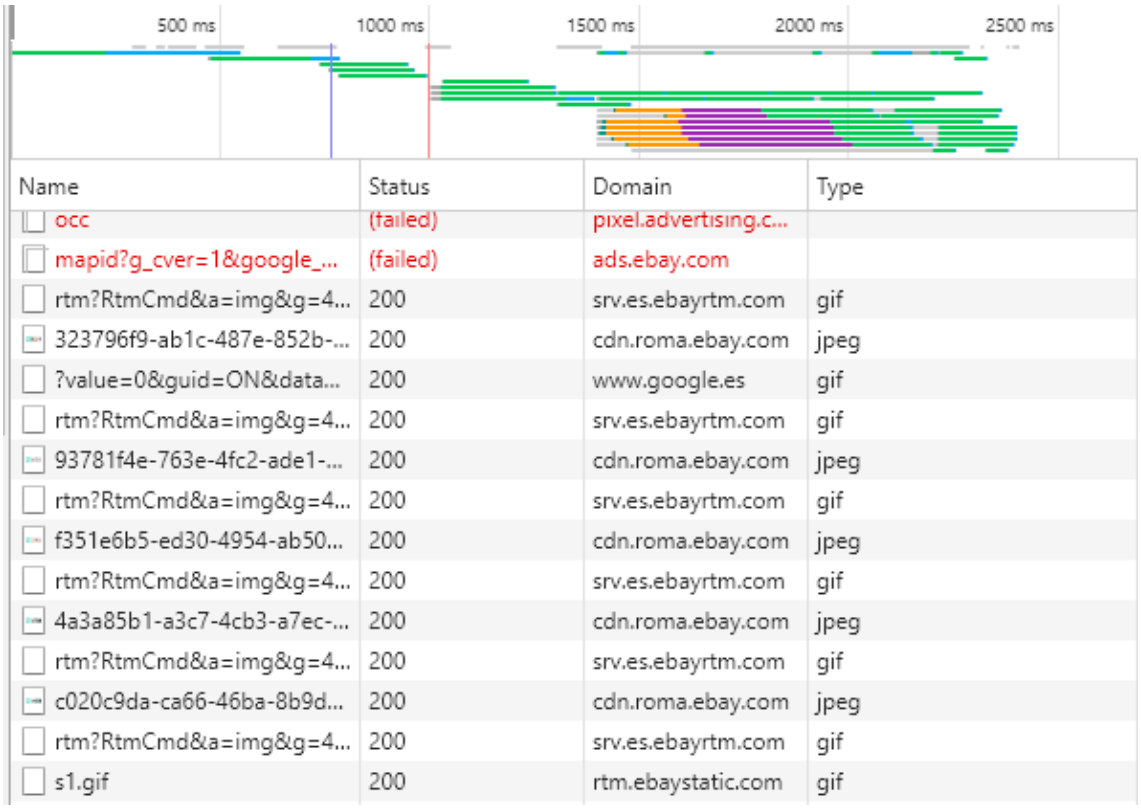
* Requiere además que los servidores soporten el protocolo de negociación mediante ALPN.

El segundo paso es observar que todos los navegadores soportan el protocolo sobre TLS. El estándar no especifica concretamente que necesite usar TLS para el protocolo, sin embargo, Google al experimentar con SPDY y utilizar la cabecera "Upgrade", funcionando más allá del puerto 80, se produjo una gran tasa de error y al probar a utilizar las mismas peticiones mediante TLS, se obtuvo una cantidad de errores muy baja en comparación a la no utilización de TLS. La otra razón por la cual se usa TLS es porque con la llegada de este nuevo protocolo se promueve encriptar todas las

comunicaciones para la seguridad y privacidad del individuo. Para ello, algunos de los navegadores web más conocidos como son Chrome o Mozilla Firefox, han decidido que solo van a implementar HTTP/2 bajo TLS.

El tercer paso es revertir algunas optimizaciones de HTTP/1.1 que no son óptimas para HTTP/2, una de ella es la concatenación, que consiste en la combinación de muchos elementos como varios CSS o JavaScript en un único fichero, para evitar hacer múltiples peticiones al servidor, a cambio de descargar un único fichero bastante grande. El problema reside cuando se realiza algún cambio, puesto que la actualización es de mucha información al ser un único fichero. Otro cambio que se debe revertir es la fragmentación, que consiste en dividir objetos en varios dominios para usar más "sockets", y puesto que HTTP/2 usa solo un "socket" y rompería esta regla, se debe revertir. Y para concluir hay una optimización llamada "spriting" que consiste en unir varias pequeñas imágenes en una más grande, que la parte que se desea usar se recortará mediante JavaScript o CSS. Sin embargo, el "spriting" queda obsoleto en HTTP/2 puesto que puede costar mucho tiempo el CSS.

Un ejemplo de fragmentación es la siguiente figura, tomada del sitio web eBay [10], donde se pueden observar la cantidad de dominios que se han de resolver:



Name	Status	Domain	Type
occ	(failed)	pixel.advertising.c...	
mapid?g_cver=1&google_...	(failed)	ads.ebay.com	
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
323796f9-ab1c-487e-852b-...	200	cdn.roma.ebay.com	jpeg
?value=0&guid=ON&data...	200	www.google.es	gif
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
93781f4e-763e-4fc2-ade1-...	200	cdn.roma.ebay.com	jpeg
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
f351e6b5-ed30-4954-ab50...	200	cdn.roma.ebay.com	jpeg
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
4a3a85b1-a3c7-4cb3-a7ec-...	200	cdn.roma.ebay.com	jpeg
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
c020c9da-ca66-46ba-8b9d...	200	cdn.roma.ebay.com	jpeg
rtm?RtmCmd&a=img&g=4...	200	srv.es.ebayrtm.com	gif
s1.gif	200	rtm.ebaystatic.com	gif

Figura 1 Dominios de una página web

Un ejemplo de “spriting” sería la siguiente figura, recogida de un artículo de devbridge, de Andrius Juskenas [9]:

Figura 2 Ejemplo de fragmentación

El cuarto paso es estudiar el contenido “third-party”, que es contenido que se obtiene de páginas externas. Para empezar HTTP/2 funciona mejor con un único “socket” y para obtener una mayor eficiencia necesita TLS. Para ello es necesario plantease si es necesario este contenido si no soportar TLS o si no tiene plan para soportar HTTP/2.

Y el quinto y último paso, es dar soporte a los clientes que no soportan HTTP/2, puesto que hay usuarios que utilizan sistemas operativos antiguos como Windows XP y puesto que estos usuarios no pueden acceder a una página en HTTP/2 con Internet Explorer. Por lo tanto, hay que tener en consideración a los clientes que quizás no puedan acceder a este servicio si se cambia a HTTP/2.

3. Conceptos relacionados

Antes de entrar en profundidad en HTTP/2, varios conceptos relacionados con HTTP/2 deben de ser entendidos, para entender las necesidades del protocolo.

HOL Blocking

El Head Of Line Blocking consiste en la saturación de la cola de salida, porque varias entradas compiten por la misma salida. Este problema puede ser causado porque en HTTP/1.1 existe un mecanismo llamado “pipelining”, que consiste en enviar muchas peticiones de una vez, pero recibiendo una respuesta después de otra hasta terminar con todas. Este mecanismo se usa porque no existe un mecanismo en HTTP/1.1 que sea capaz de pedir varios objetos simultáneamente. Pero si la conexión queda atascada por algún problema en alguna petición o respuesta, se tarda bastante tiempo en cargar la página. Para evitar esto, se utilizan 6 conexiones paralelas, pero es posible que cada una de estas conexiones sufran HOL Blocking. Además, realizan un mal uso de TCP, al usar 6 conexiones.

ALT-SVC

Alt-svc [15] es una cabecera de HTTP que informa el servidor de origen sobre la disponibilidad de servicios alternativos a los clientes.

Por ejemplo:

Alt-Svc: h2=”:8000”

Indica que el mismo host bajo el puerto 8000 está escuchando en HTTP/2.

Otro ejemplo sería:

Alt-Svc: h2= ”new.example.org:80”

Indica que en el host new.example.org en el puerto 80, está escuchando en HTTP/2.

SSL/TLS

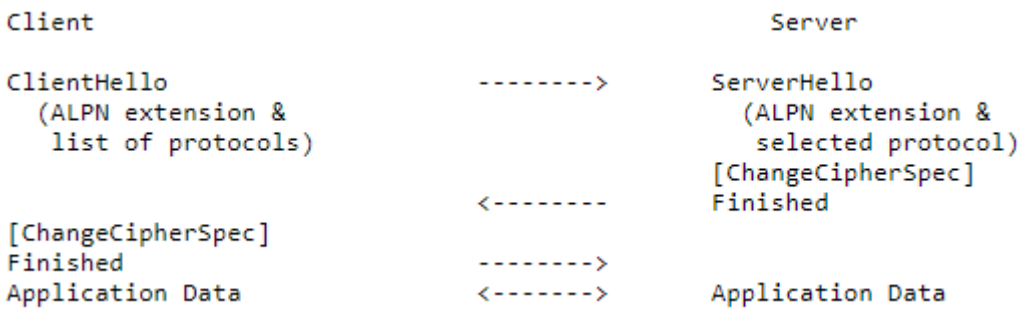
SSL es una capa de sockets seguros, que permiten tener una conexión segura a Internet, que tiene como finalidad proteger toda la información que se desean intercambiar los sistemas para evitar que circule esa información como texto plano.

TLS es la versión más reciente de SSL, se define como Seguridad de la Capa de Transporte, tiene la misma finalidad que SSL, que es proteger una conexión segura mediante protocolos criptográficos, y la diferencia es que es la versión que derivó de SSL

ALPN

ALPN [16] es una extensión de TLS que sirve para negociar en la capa de aplicación, el protocolo que se utilizará, dentro de "handshake" de TLS, permitiendo a la capa de aplicación negociar el protocolo usado en la conexión TLS.

Un ejemplo de cómo funciona sería el siguiente:



El cliente usaría ALPN y le proporcionaría una serie de protocolos, de los cuales el servidor seleccionaría el protocolo, cambiaría su cifrado al deseado y se lo enviaría al cliente, con esto se terminaría la negociación del protocolo y se enviaría confirmación por parte del cliente para empezar a mandar datos en la capa de Aplicación.

Resumen HTTP

Para más detalles sobre el estándar, se puede consultar el RFC 2616 [17]

Métodos en Petición

GET

Pide un recurso especificado, se considera idempotente, por lo que se puede almacenar en cache y se puede repetir múltiples veces, puesto que no altera el recurso. Puede ser seguido de diferentes cabeceras.

HEAD

Pide una respuesta del recurso especificado, pero esta vez no se muestra el cuerpo de la petición, solo las cabeceras. Es idempotente como GET.

POST

Altera el recurso accedido, para ello envía datos que serán incluidos en la petición para ser procesados después. Se considera un método no idempotente, puesto que modifica el objeto.

DELETE

Borra el recurso especificado

PUT

Carga el recurso especificado en el servidor

Cabeceras

Son metadatos que se envían en peticiones o respuestas para proporcionar información necesaria para la correcta interpretación de los datos intercambiados sin retocar el protocolo. Cada cabecera tiene un valor y seguida tiene un retorno de carro seguido de un salto de línea. Para finalizar la lista de cabeceras se usa una línea en blanco.

Códigos de Respuesta

Es un número que indica el resultado de la petición, cada número tiene un contenido asociado que aporta más información de lo sucedido.

-Códigos de 100 a 199: Respuestas informativas. Petición recibida y procesándose.

-Códigos de 200 a 299: Respuestas correctas. Petición procesada correctamente

-Códigos de 300 a 399: Respuestas de redirección. Acción necesaria para finalizar la petición.

-Códigos de 400 a 499: Errores causados por el cliente. Error en el procesado de la petición a causa del cliente

-Códigos de 500 a 599: Errores causados por el servidor. Error en el procesado de la petición a causa del servidor.

4. Características de HTTP/2

HTTP/2 es un protocolo que se encuentra en la capa de aplicación según el modelo OSI, además, este protocolo está sobre una conexión TCP. HTTP/2 utiliza los mismos esquemas que HTTP/1.1 en cuanto a http y https, redireccionando a los puertos 80, para http y 443 para https.

El protocolo tiene dos identificadores, el primero de ellos se trata de "h2" y es un identificador usado cuando HTTP/2 usa TLS y es codificado dentro de un identificador de protocolo ALPN con la secuencia de dos octetos: 0x68 y 0x32, que se corresponde con en el código ASCII a los caracteres h2. El segundo se trata de "h2c" que identifica a HTTP/2 como un protocolo que no usa TLS.

4.1 Inicio de conexión en HTTP/2

Las URLS http:// y https:// ya existentes no pueden ser cambiadas, por lo que HTTP/2 ha de ser actualizado mediante los esquemas ya existentes en HTTP/1.x mediante la cabecera "Upgrade", este método queda un poco obsoleto en cuanto a eficiencia temporal, puesto que el protocolo trata de mitigar el tiempo necesario para que cargue la página, y esta negociación consume un valioso tiempo.

Conexión desde http sin conocimiento

Cuando un cliente realiza una petición http sin conocer si soporta el servidor HTTP/2, el cliente usa la cabecera de HTTP llamada "Upgrade" con el contenido "h2c". Además de esto, debe incluir exactamente una línea de cabecera HTTP2-Settings. Una posible petición de "upgrade" seria la siguiente:

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

Figura 3 Petición "Upgrade" en http

Si el servidor no soporta HTTP/2, le contestaría con un HTTP/1.1 200 OK:

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
```

Figura 4 Respuesta a "Upgrade" 1/2

Pero, si el servidor soportase HTTP/2 la respuesta cambiaría a un HTTP/1.1 101 y seria la siguiente:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

[ HTTP/2 connection ...
```

Figura 5 Respuesta a "Upgrade" 2/2

En HTTP/2, es necesario que cada interesado mande un prólogo con el fin de confirmar el protocolo en uso y establecer una conexión con las opciones iniciales que se han decidido adoptar.

Para ello, una vez el cliente ha recibido la respuesta del servidor 101, el cliente debe mandar un prólogo que contiene una trama con las opciones.

El prólogo del cliente siempre empieza con una secuencia de 24 octetos que se repiten siempre y son seguidos de una trama de opciones que puede estar vacío. En notación hexadecimal, el prólogo es el siguiente:

0x505249202a20485454502f322e300d0a0d0a534d0d0a0d0a
Y seria: "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n"

La primera trama enviado por el servidor debe de ser un prólogo de conexión del servidor que consiste en una trama de opciones que muy probablemente sea vacío.

Conexión desde https sin conocimiento

Un cliente que realiza una petición https sin conocer si soporta el protocolo HTTP/2, usa TLS con la extensión de negociación en la capa de aplicación, conocido como ALPN, el cliente realiza una petición de "upgrade" como en el caso anterior, pero en la cabecera "Upgrade" utiliza la identificación "h2" que implica el uso de TLS. Una vez se ha completado la negociación de TLS, tanto el cliente como el servidor envían el prólogo descrito anteriormente.

Conexión HTTP/2 con conocimiento previo

Puesto que un cliente puede conocer que un servidor puede soportar HTTP/2 por otros métodos, como por ejemplo ALT-SVC, que describe un mecanismo para advertir de que el servidor soporta HTTP/2, descrito en la página 22.

Para ello, una vez el cliente sabe que el servidor soporta HTP/2, envía el prólogo de conexión y puede que empiece a enviar tramas de HTTP/2 al servidor. Igualmente, el servidor debe enviar su prólogo de conexión al cliente.

De esta forma, se consigue mitigar el tiempo empleado en la comunicación TCP y por lo tanto aumentar la eficiencia.

4.2 Tramas y datos

HTTP/2 es un protocolo binario para conseguir hacer que las tramas sean más sencillas, puesto que determinar dónde empieza y donde acaba una trama es una cosa realmente complicada en los protocolos de texto, como es HTTP/1.1, con esto se pretende que la implementación para la lectura del protocolo sea más simple y no confusa.

Por lo tanto, las tramas que envía HTTP/2 son binarias, pero todas comienzan de la misma forma. La estructura de las tramas se puede observar en la siguiente figura:

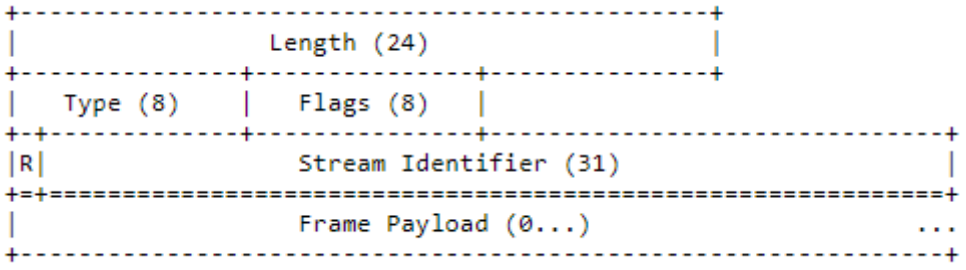


Figura 6 Trama HTTP/2

Los campos de la trama de HTTP/2 serían los siguientes:

- Length: Determina la longitud del “payload” y está representado mediante un entero de 24 bits sin signo. Los primeros 9 octetos de la cabecera de la trama no están incluidos. El tamaño máximo predefinido por el estándar sin cambiar el campo SETTINGS_MAX_FRAME_SIZE es 2^{14} octetos y el máximo cambiándolo es de 2^{24} -1.

- Type: 8 bits que especifican el tipo de trama.

- Flags: Campo de 8 bits reservado para funciones booleanas específicas de cada tipo de trama.

-R: Campo de 1 bit reservado.

-Stream identifier: Determina a qué flujo pertenece la trama y está representado mediante un entero de 31 bits sin signo.

-Frame payload: Es variable y depende de cada tipo, es el contenido de la trama y su longitud está determinada por el primer campo, “Length”.

Existen 10 tipos de trama especificados en HTTP/2 y se explican en el Anexo A de este documento.

4.3 Flujos

Es importante definir lo que es un flujo para entender qué es y cómo funciona la multiplexación de flujos. Una trama está asociada a un flujo, como se ha descrito anteriormente. Un flujo es una asociación lógica de distintas tramas independientes y bidireccionales intercambiadas entre un servidor y un cliente en una conexión HTTP/2. Cuando un cliente pretende realizar una petición, inicia un nuevo flujo y el servidor responderá en el mismo flujo.

Puesto que una conexión puede contener muchos flujos simultáneos, la multiplexación de los flujos consiste en que los paquetes de distintos flujos en la misma conexión se mezclan, puesto que los receptores procesan las tramas en el orden que son recibidas. Es por ello, que el orden en el que se envía cada flujo es significativo.

Los flujos pueden crearse y ser usados por el cliente o el servidor, dando la posibilidad de ser cerrados en cualquier momento.

Como podemos observar los mensajes que se transportan son muy simples, puesto que como mínimo se envían tramas de cabeceras y puede obtener adicionalmente tramas de datos, siguiendo así la semántica de HTTP/1.1 donde las cabeceras estaban al principio de una trama y los datos separados por una línea en blanco.

Al contrario que HTTP/1.1, el protocolo HTTP/2 establece un control de flujo distinto, mientras que en su versión 1.1, el servidor enviaba datos tan rápido como el cliente los consumía, en esta nueva versión se da la posibilidad al cliente o al servidor de establecer el ritmo del envío de bytes. Para ello se envía esta información en la trama llamada WINDOW_UPDATE, que establece la cantidad de bytes que desea recibir el receptor. A medida que el receptor reciba y consuma datos, se envía otra trama WINDOW_UPDATE para indicar su nuevo número de bytes que puede recibir.

Esta implementación de control de flujo sirve para que unos flujos no usen todo el ancho de banda, permitiendo así que otros flujos que requieren menor ancho de banda, no resulten ahogados. La contraparte de este mecanismo es que no puede ser desactivado, pero usando un valor de 2^{31} -1 lo deshabilita para ficheros menores de 2 GB.

Además del control de flujo, necesitamos otra herramienta para determinar qué flujo es importante puesto que puede haber dependencias, y por lo tanto darle prioridad es primordial, para ello HTTP/2 establece prioridades en sus flujos. Utilizando la trama PRIORITY, un cliente puede indicar al servidor que un flujo, es dependiente de otro flujo. Esto permite al cliente construir un árbol de pesos, que permite obtener a los flujos prioritarios antes que los menos prioritarios, estos pesos pueden ser cambiados

dinámicamente, por lo que permite a un usuario que navega sobre una página con miles de objetos, que se cambien las prioridades de los objetos a medida que el usuario haga "scroll". Además de esta priorización indicando las dependencias entre objetos, existe otro valor llamado peso que es un entero cuyo valor oscila entre 1 y 256 (ambos incluidos). Cuando los objetos dependientes se han recibido, se asignan valores a los restantes, y el cliente realiza un esfuerzo para cada objeto restante, proporcional al total de pesos.

Por ejemplo, si los objetos restantes fueran:

-Imagen (peso = 5)

-Imagen poco critica (peso =2)

-Imagen necesaria (peso=10)

-JS poco crítico (peso =8)

El esfuerzo total seria 5+2+10+8= 25.

Por lo tanto, el cliente haría un esfuerzo de 5/25 para obtener la primera imagen, es decir, usaría una quinta parte de su esfuerzo para obtenerla.

Un ejemplo de cómo funciona HTTP sin multiplexación sería la siguiente figura:

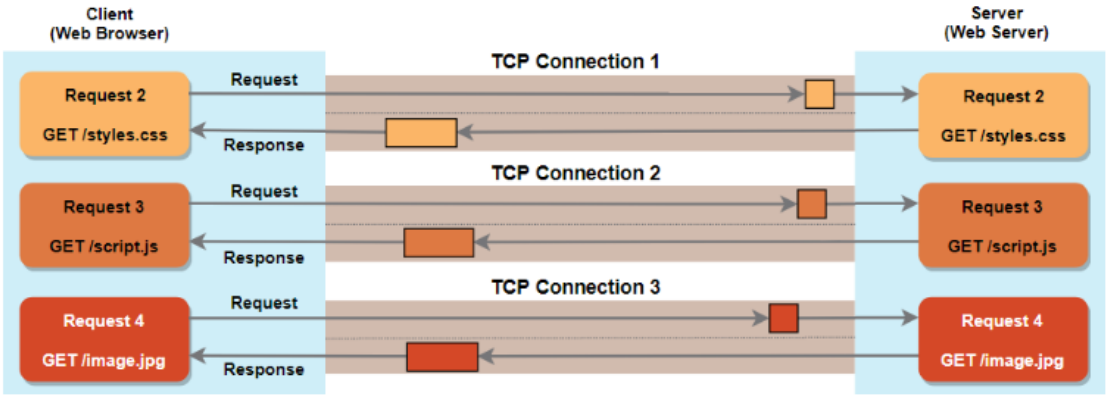


Figura 7 Ejemplo de conexión sin multiplexación

Como se puede observar hay que esperar el tiempo de respuesta para obtener cada objeto requerido, siempre se mantiene un orden y es más ineficiente. Y para compararlo con la multiplexación, aquí se adjunta la siguiente figura:

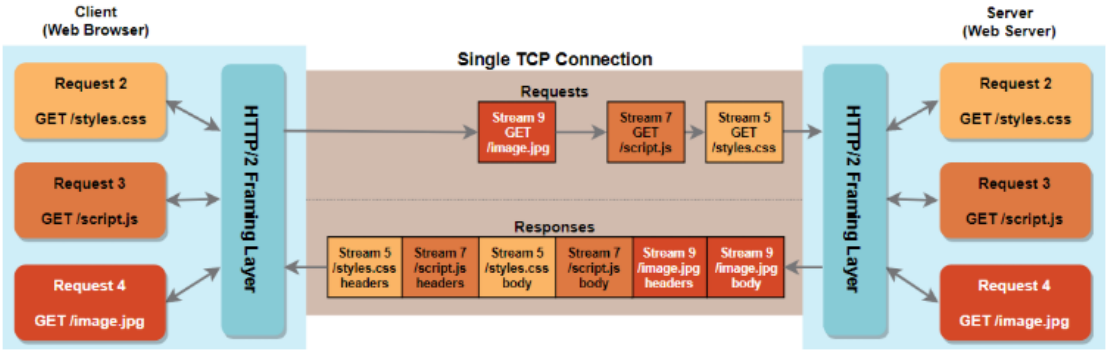


Figura 8 Ejemplo de conexión HTTP con multiplexación

Como se observa, en una misma conexión, se solicitan 3 objetos, se realiza un árbol para obtener los objetos más prioritarios antes y se intercalan usando la multiplexación.

4.4 Compresión de cabeceras

Cuando se observa la estructura de los mensajes en HTTP/2 al igual que en HTTP/1.1, muchos de los bytes de las peticiones son iguales, puesto que algunas peticiones se repiten como la cabecera user-agent, que siempre será el mismo navegador web en una misma sesión, o la cabecera accept-language que es el idioma que acepta. Esta circunstancia puede provocar un pequeño retraso en el mejor de los casos, pero en una situación donde el ancho de banda es limitado, puede provocar grandes retrasos. Aproximadamente, el 85% de los bytes enviados se pueden comprimir.

Para comprimir las peticiones se usa HPACK que usa dos técnicas muy potentes, una de ellas es la compresión de Huffman para codificar con menos bytes, las cabeceras más utilizadas, y la segunda es requerir que los clientes y servidores mantengan una lista actualizada de los índices que referencian a las cabeceras para transmitir eficientemente los valores de las cabeceras.

Un ejemplo para comprender mejor el funcionamiento de la compresión de cabeceras es la siguiente tabla:

Tabla 2 Compresión de cabeceras HPACK 1/2

CABECERA	DATO	INDICE TABLA	¿TRANSMITIDA CABECERA?
:method	GET	2	SI
:host	ejemplo.com	62	SI
:path	/recurso	63	SI
user-agent	Mozilla/5.0	64	SI

Tabla 3 Compresión de cabeceras HPACK 2/2

CABECERA	DATO	INDICE TABLA	¿TRANSMITIDA CABECERA?
:method	GET	2	NO
:host	ejemplo.com	62	NO
:path	/recurso_2	65	SI
user-agent	Mozilla/5.0	64	NO

Como se puede observar solo se transmitiría la nueva cabecera :path y se le asignaría el valor en la tabla 65. Las otras cabeceras son transmitidas mediante los índices de las tablas.

4.5 Server push

Para mejorar el rendimiento de un objeto en particular, la mejor forma es tenerlo en la cache antes de que el navegador lo pida. Esta es la meta de la característica “Push” de los servidores HTTTP/2, que consiste en la capacidad de enviar un objeto a un cliente cada cierto tiempo, que probablemente se solicitará en un futuro próximo, permitiendo al servidor enviar a los clientes arbitrariamente los objetos.

Cuando el servidor decide enviar un objeto mediante “push”, se construye una trama PUSH_PROMISE que poseen los siguientes atributos importantes:

- El identificador de la trama PUSH_PROMISE es el identificador de la petición cuya respuesta está asociada. Las respuestas están siempre relacionadas con alguna petición y además en la trama PUSH_PROMISE se indica el identificador de flujo donde estará la respuesta.

- La trama PUSH_PROMISE tiene la cabecera que dictamina que se va a enviar si el cliente pide el objeto.

- El objeto debe de poderse almacenar en la cache.

- El método solicitado debe de ser seguro, por lo tanto, el método POST no puede enviarse.

- Idealmente la trama PUSH_PROMISE debe de ser enviada antes de recibir tramas de datos, puesto que si el servidor envía todo el HTML antes de que la trama PUSH_PROMISE se envíe, el cliente, probablemente habrá pedido el objeto.

Cuando el cliente está insatisfecho con algún elemento de alguna trama de PUSH_PROMISE, se puede reiniciar el nuevo flujo mediante el flujo RST_STREAM o enviar una trama PROTOCOL_ERROR, en función de la razón del rechazo. Una causa común es que el objeto ya está en cache.

Dependiendo de la aplicación, decidir qué objetos enviar por “push” es bastante complejo. Para ello se tienen que tener en cuenta los siguientes aspectos:

- Las probabilidades de que el objeto esté ya en la cache del navegador.
- La prioridad del objeto para el cliente.
- El ancho de banda del cliente que puede mermar la capacidad de “push”.

Si se escogen correctamente los objetos puede ayudar al rendimiento, pero si se escogen mal, puede derivar en el efecto contrario.

5. Rendimiento de HTTP/2

5.1 Rendimiento HTTP/2 en navegadores web

HTTP/2 es un protocolo muy reciente, por lo que las implementaciones pueden variar, tanto por parte del cliente, como algunas características en los servidores, como la capacidad del server push o establecer el árbol de dependencias correctamente. Es por ello una realidad que una misma página web puede tardar más o menos en ser cargada usando la misma conexión, pero con diferentes navegadores web.

Un ejemplo de estas diferencias de tiempo se puede observar mediante un estudio que se realizó por parte del equipo de Akamai que comparó HTTP/1.1 y HTTP/2 en diferentes navegadores para comprobar que hay diferentes rendimientos. Esta investigación involucró billones de monitorizaciones de dispositivos reales bajo condiciones reales de red. El estudio concluyó de la siguiente forma:

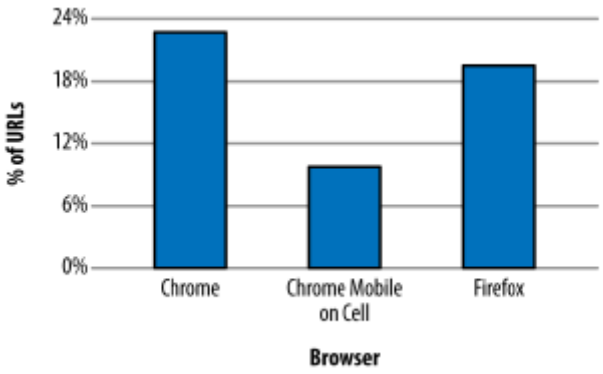


Figura 9 Porcentaje de páginas web que mejoran con HTTP/2

La figura anterior muestra la cantidad de URLS que mejoraron su rendimiento significativamente. El estudio muestra que un 20% de urls aproximadamente en los navegadores de ordenadores de sobremesa fueron más rápidos en HTTP/2 que en HTTP/1.1. Sin embargo, este estudio también mostró los siguientes resultados:

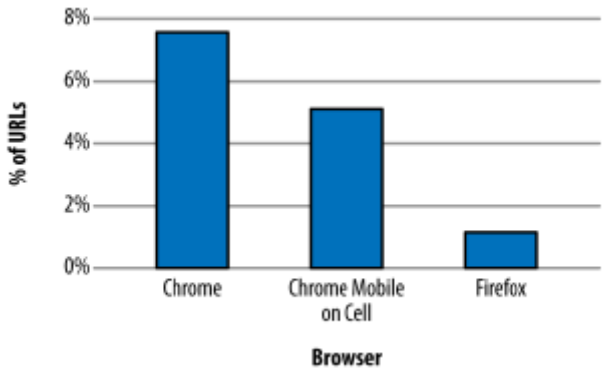


Figura 10 Porcentaje de páginas web que empeoran con HTTP/2

La figura anterior muestra la cantidad de urls que obtienen un peor rendimiento en HTTP/2 que en HTTP/1.1. Por lo tanto, es necesario concluir que algunos navegadores como Chrome tienen un número mayor de urls que son mejoradas con HTTP/2 que HTTP/1.1, pero que también tienen casi un 8% de urls que son peores que en HTTP/1.1

Estas diferencias de tiempos se deben mayoritariamente al establecimiento de dependencias entre los navegadores web, que puesto que son distintos se obtienen resultados diferentes. Por ejemplo, una página web puede verse ralentizada si existe un css o un archivo JavaScript que no tenga alta prioridad y haga que la carga de la página web sea más lenta. Es importante remarcar que Android tiene el peor rendimiento, esto puede ser debido a que muchas páginas que se usan habitualmente en los ordenadores no tengan utilidad en los móviles, como puede ser Twitter o Facebook, que usan sus propias aplicaciones para móviles.

5.2 Rendimiento HTTP/2 vs HTTP/1.1

Para realizar un estudio con páginas webs actuales que tienen un gran número de peticiones actualmente he planteado 6 escenarios distintos, que son los siguientes:

Tabla 4 Test realizados HTTP/2

Escenario	Ancho de banda	Casos	Método
1 - ADSL	5 Mbps	50/100/150 ms	Webpagetest
2 - 3G	1,6 Mbps	150/300/400 ms	Webpagetest
3 - 4G	9 Mbps	70/120/170 ms	Webpagetest
4 - Fibra Óptica	300 Mbps	20/50/80 ms	Webpagetest
5 - Fibra Óptica	300 Mbps		Análisis Real
6 - Akamai		HTTP/2 vs HTTP/1.1	Demo
7 - Akamai		Push vs No Push	Demo

Los escenarios simulados mediante Webpagetest intentan recrear el rendimiento que sufre HTTP/2 ante la congestión que puede haber en la red, para ello se disponen

de 3 escenarios, uno con una latencia baja, otro con una latencia media, y el ultimo con una latencia alta. Es importante remarcar que las conexiones son simuladas y pretenden recrear las reales, estableciendo un ancho de banda estándar, con unas latencias aproximadas y lo más realistas posibles. Para obtener datos más reales se analizarán los escenarios 5 y 6 que se realizarán mediante una conexión real.

El navegador usado en este análisis va a ser Chrome, puesto que es el que mayor número de páginas es beneficiado, pero también el más penalizado, además la simplicidad para desactivar HTTP/2, que se explica en el apartado 4.2.5, es de gran utilidad para analizar correctamente el protocolo.

Las páginas web analizadas son: YouTube, Twitter y Facebook. Tres páginas web de las más visitadas en el mundo y que emplean HTTP/2.

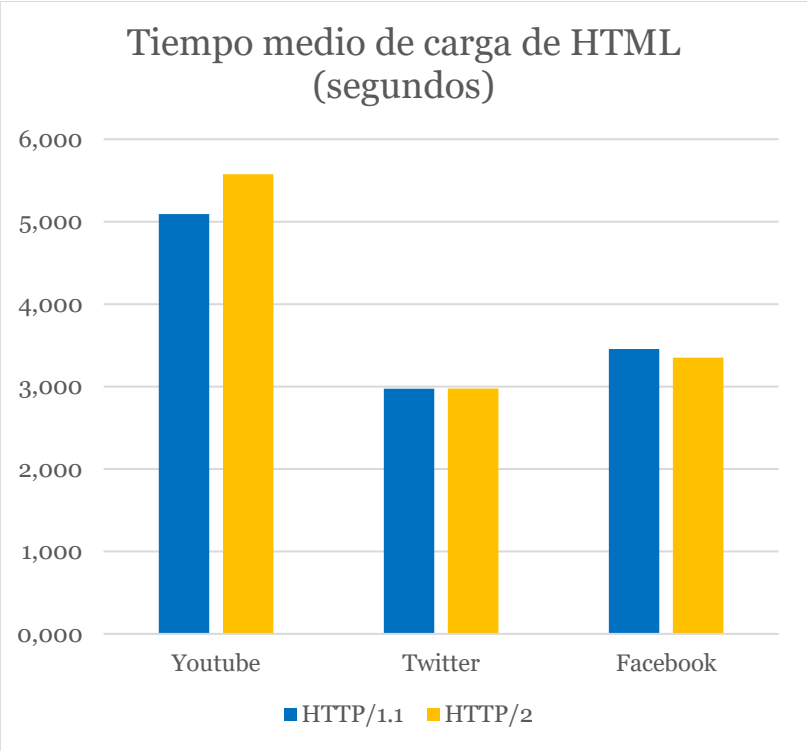
Es importante destacar que los resultados obtenidos de Webpagetest es la mediana de los 9 test realizados en cada página web con las mismas condiciones, puesto que pueden existir medias muy ruidosas que distorsionarían los datos obtenidos. Para determinar además qué test es el adecuado, se ha escogido el test de la mediana en el tiempo medio de carga, que es el tiempo que transcurre entre que se empieza a descargar una página y el tiempo en el que se visualiza en la pantalla.

5.2.1 Pruebas simuladas

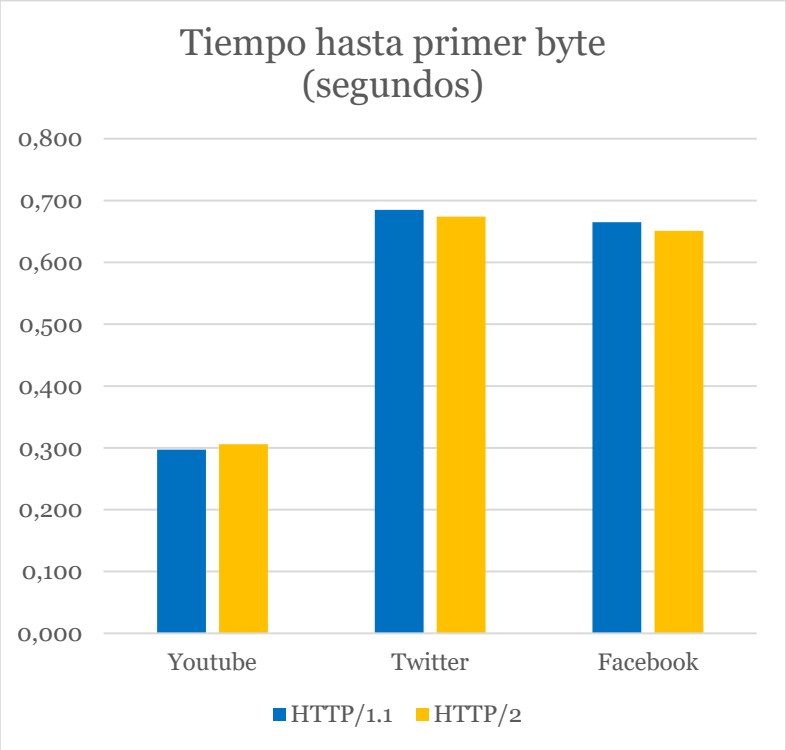
ADSL

Para realizar este estudio, en la herramienta Webpagetest, se estableció un ancho de banda de 5Mbps de bajada y 1,6 Mbps de subida, como mejor caso, se estableció una latencia de 50 ms, como caso medio una latencia de 100 ms y como peor caso una latencia de 150 ms, para ver cómo se obtienen los resultados, consultar [Anexo B].

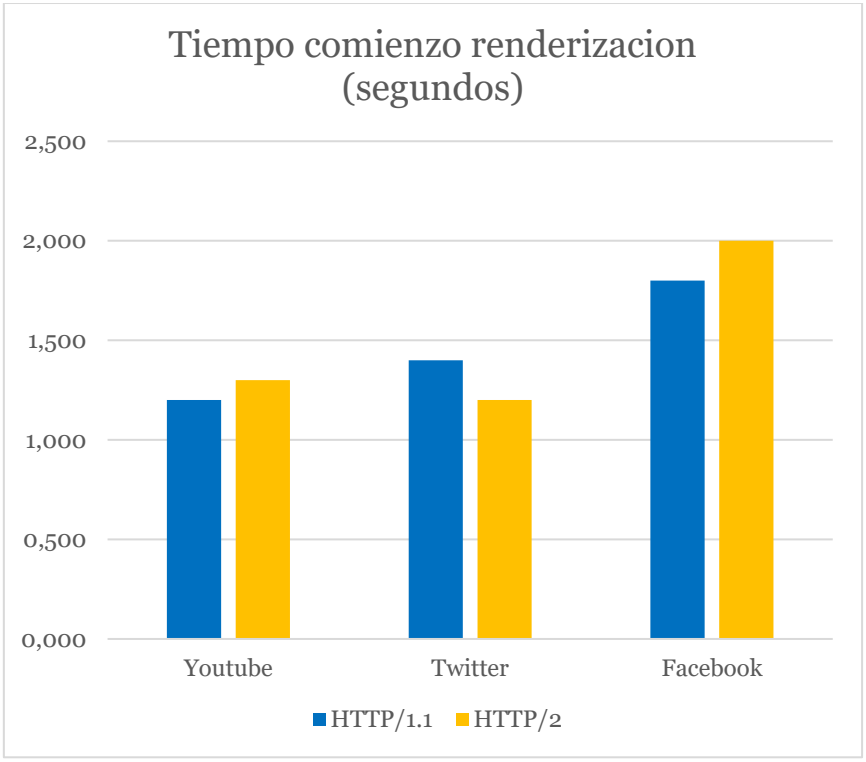
Los resultados obtenidos fueron los siguientes en el mejor caso (es decir, con la latencia de 50 ms):



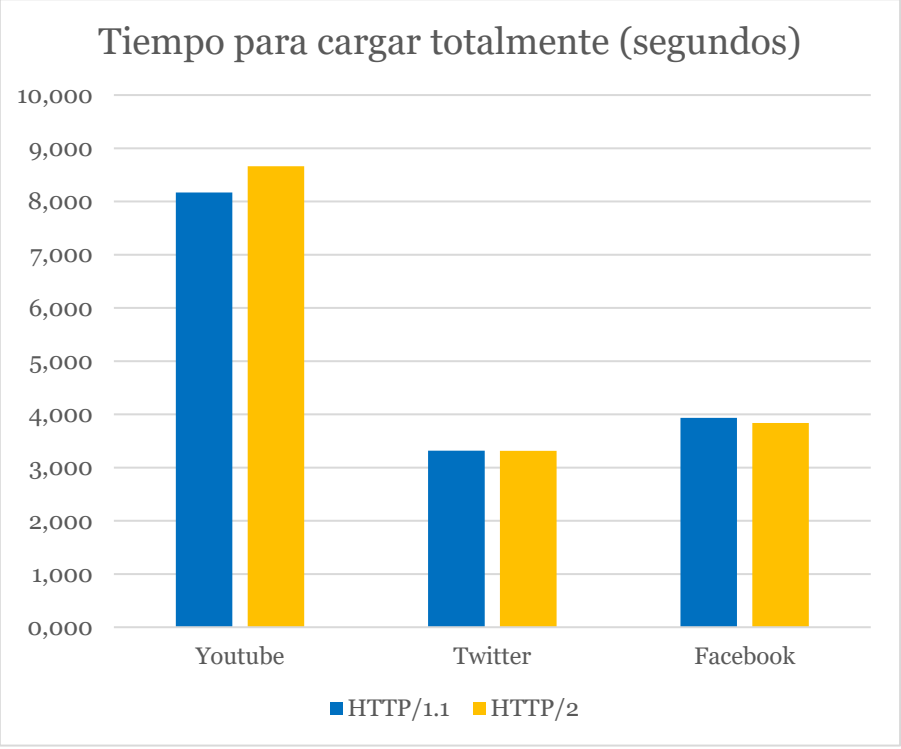
Como se puede observar en la figura anterior el tiempo de carga de las páginas web es significativamente peor en YouTube en HTTP/2, mientras que en Twitter es muy similar y en Facebook es menor en HTTP/2.



En cuanto al tiempo para el primer byte, las 3 páginas tienen un resultado similar.

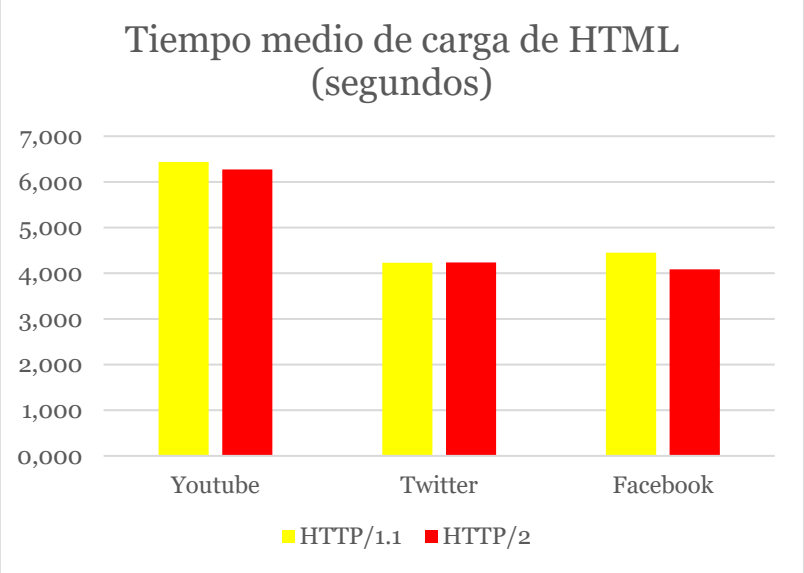


Sin embargo, en el tiempo del comienzo de la renderización de las páginas web, que es el momento en el cual una página empieza a mostrarse progresivamente al usuario en la pantalla, es disperso, en HTTP/2 es más rápido en Twitter, pero sin embargo es más lento en YouTube y Facebook.

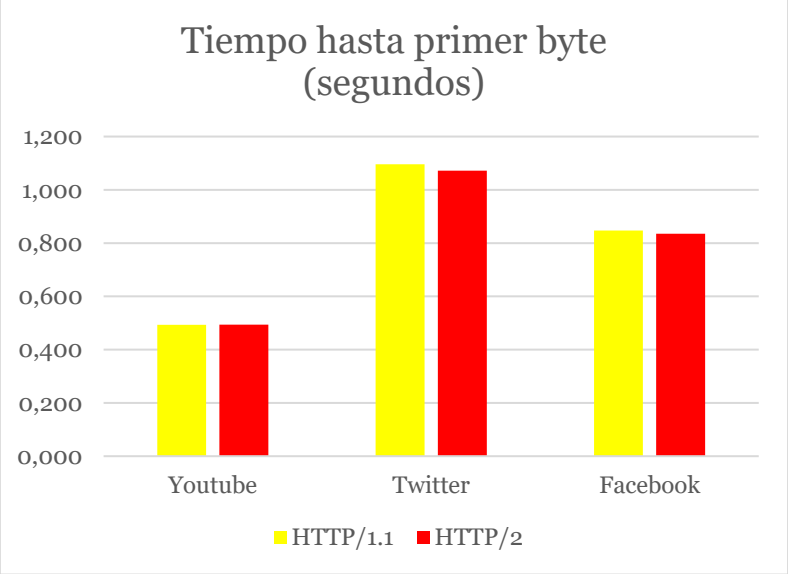


Para concluir con el análisis de ADSL con 50 ms de latencia, se analizará el tiempo para cargar al completo la página web, no solo el documento, sino todos los objetos, y con ello obtenemos resultados parecidos al tiempo para cargar la página, en Twitter apenas hay diferencia, en YouTube es peor HTTP/2 y en Facebook es ligeramente más rápido en HTTP/2.

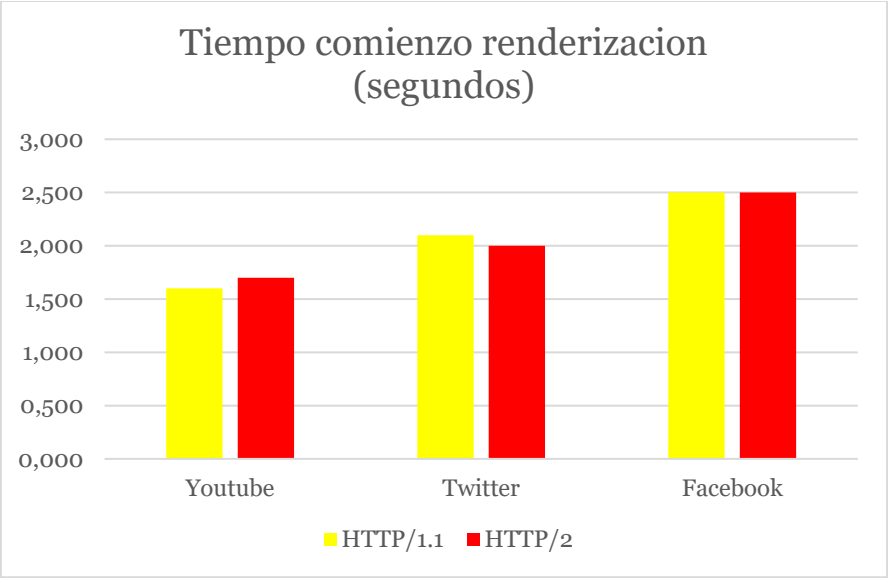
En cuanto al caso intermedio (100 ms) los resultados obtenidos fueron:



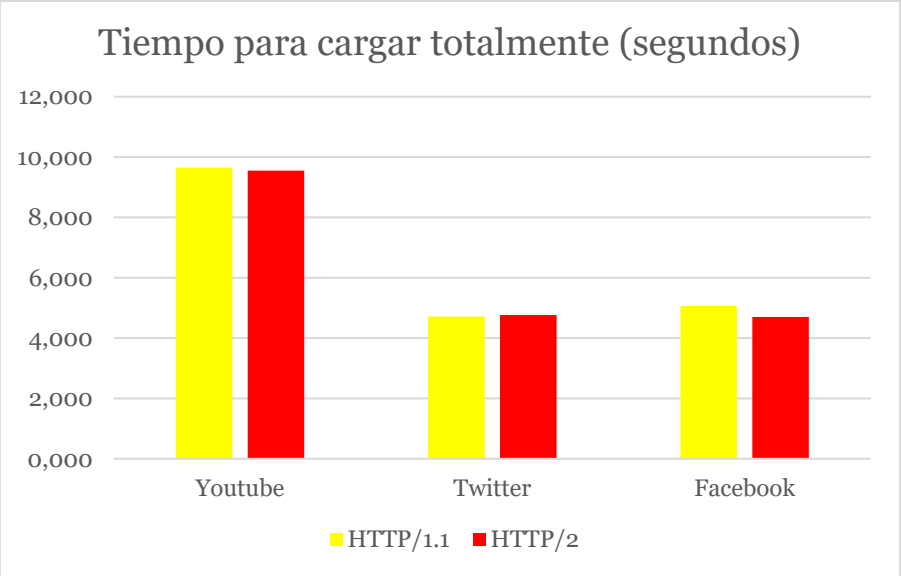
El tiempo de carga sigue siendo parecido en Twitter, pero sin embargo YouTube mejora sus prestaciones, siendo más rápido en HTTP/2 como lo es Facebook.



En cuanto al tiempo hasta el primer byte, son todos muy parecidos.

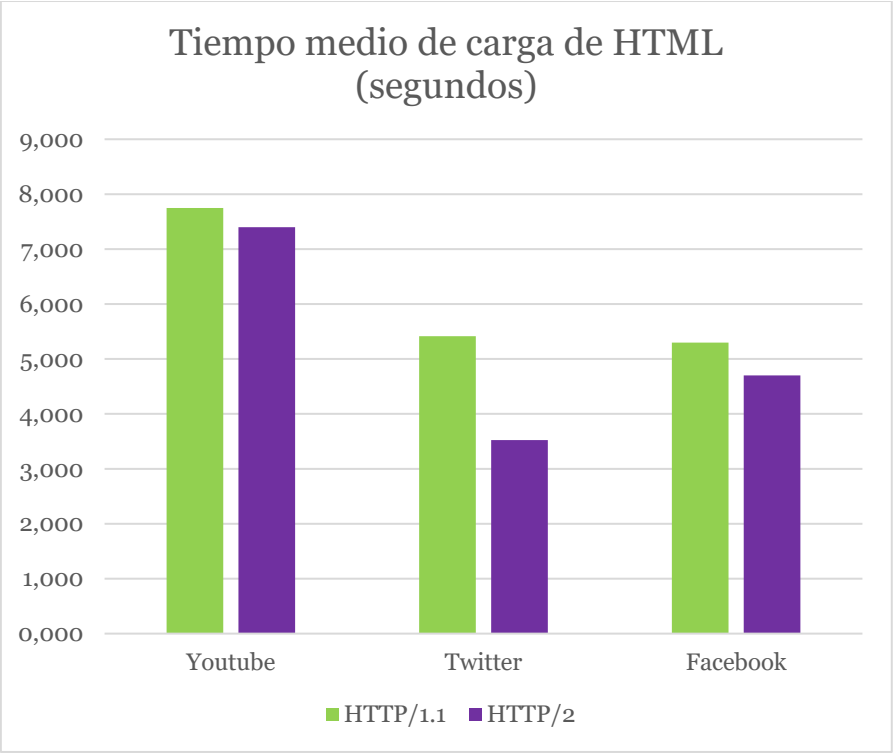


En el tiempo de comienzo de renderización Facebook, mejora respecto al mejor caso y el resto se mantiene en un estado similar.

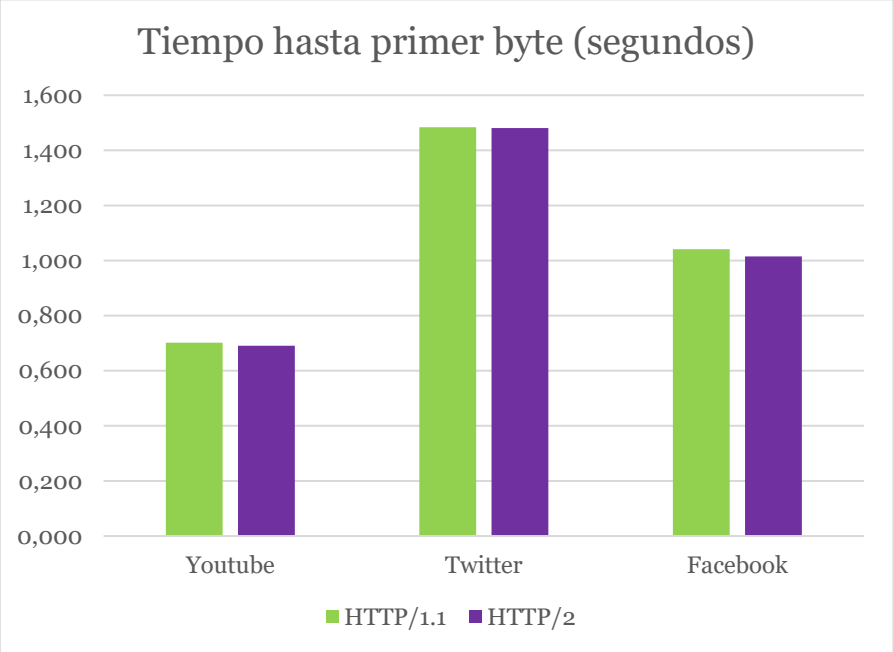


Y para finalizar el caso intermedio, se observa que los tiempos son muy similares excepto en Facebook que mejora sustancialmente en HTTP/2 a HTTTP/1.1.

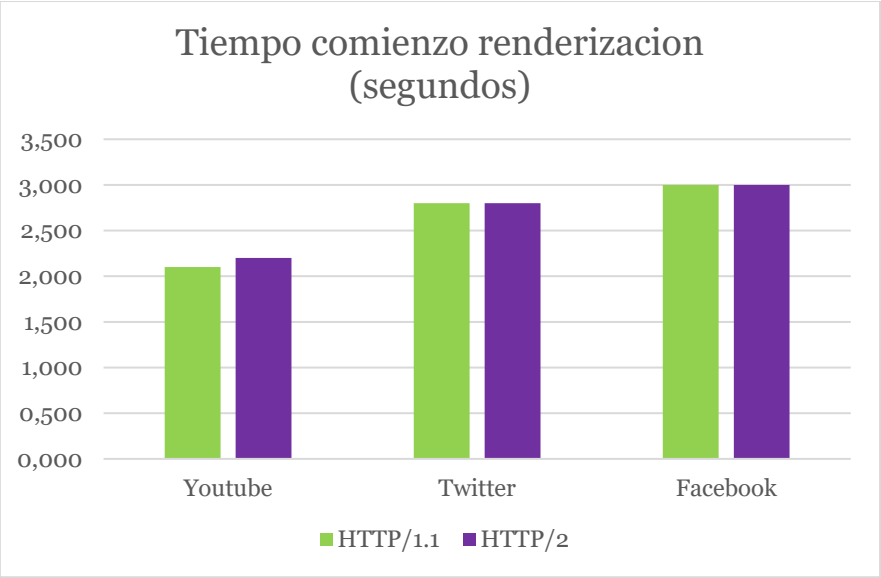
Y por último queda el peor caso (150 ms):



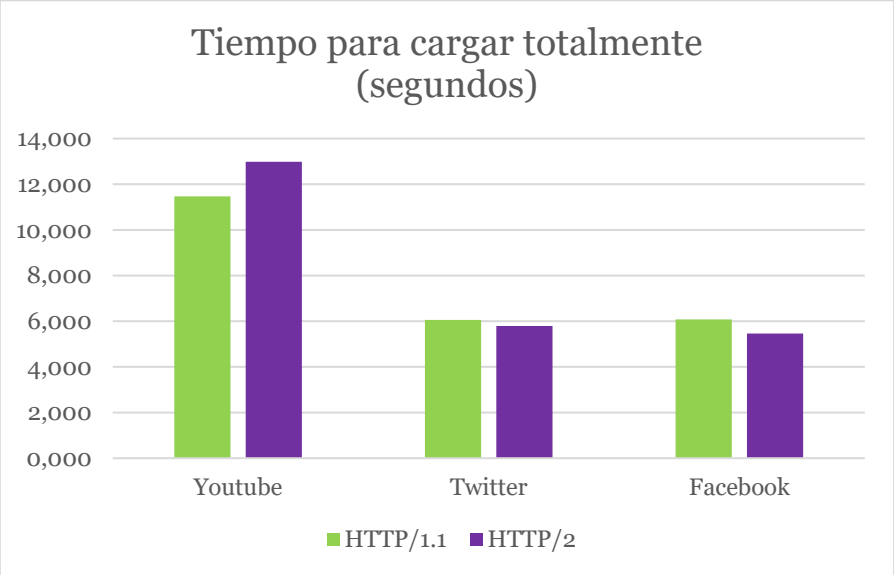
En este caso, en las 3 páginas web es mejor HTTP/2.



El tiempo hasta el primer byte es similar en todos.



El tiempo del comienzo de renderización es similar en todos, aunque un poco peor en YouTube en HTTP/2.

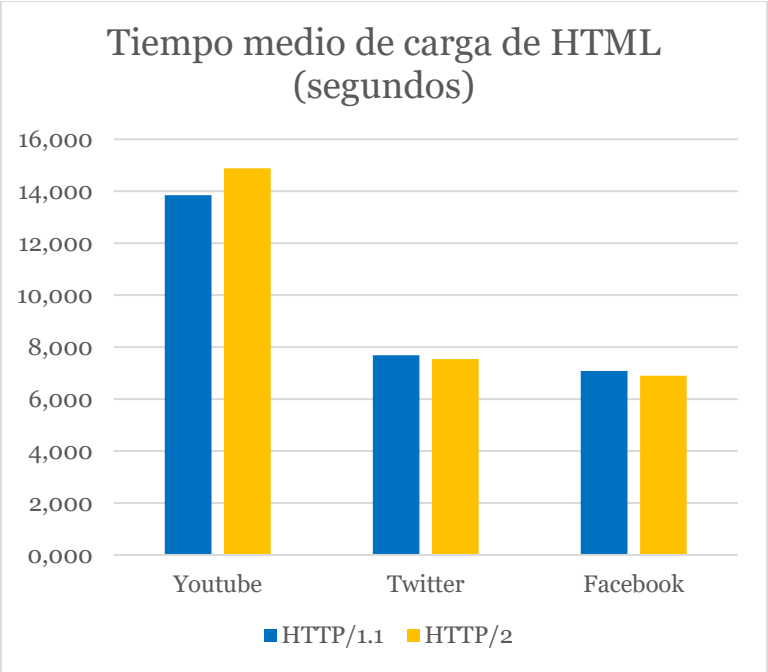


Finalmente, el tiempo para cargar totalmente la página, destaca que YouTube tarda más con HTTP/2 que con HTTP/1.1, mientras que Facebook tarda menos, al igual que Twitter, pero es mejor tiempo en Facebook.

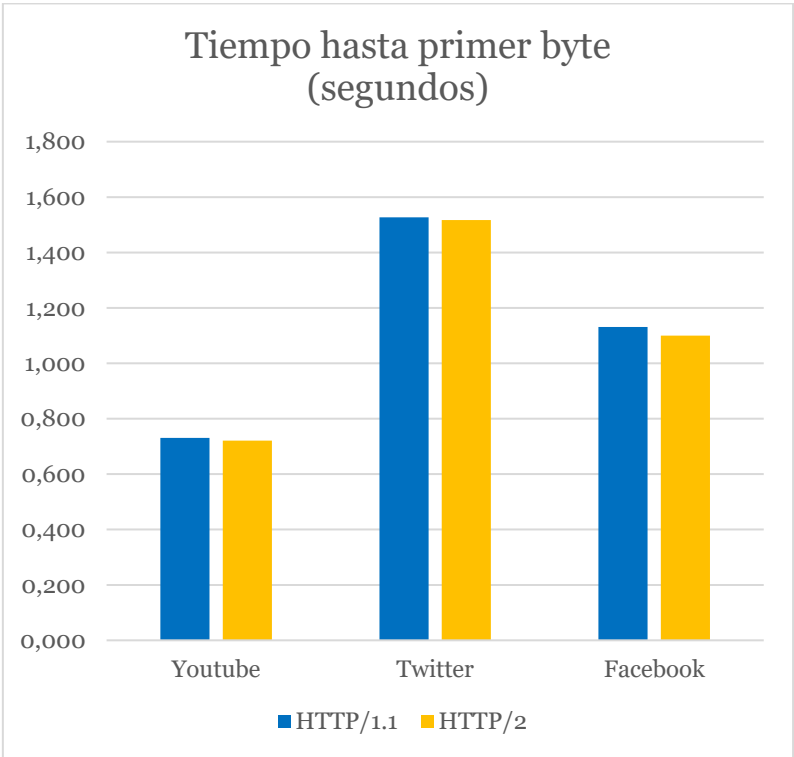
3G

Para realizar este estudio, en la herramienta Webpagetest, se estableció un ancho de banda de 1,6Mbps de bajada y 768 Kbps de subida, como mejor caso, se estableció una latencia de 150 ms, como caso medio una latencia de 300 ms y como peor caso una latencia de 400 ms.

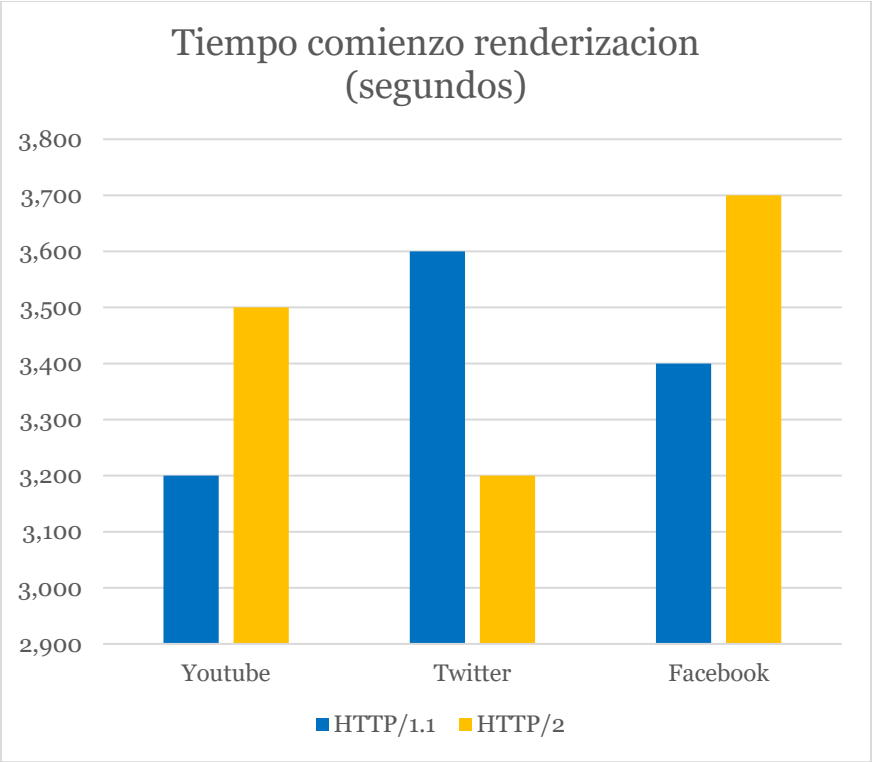
Los resultados obtenidos fueron los siguientes en el mejor caso (150 ms):



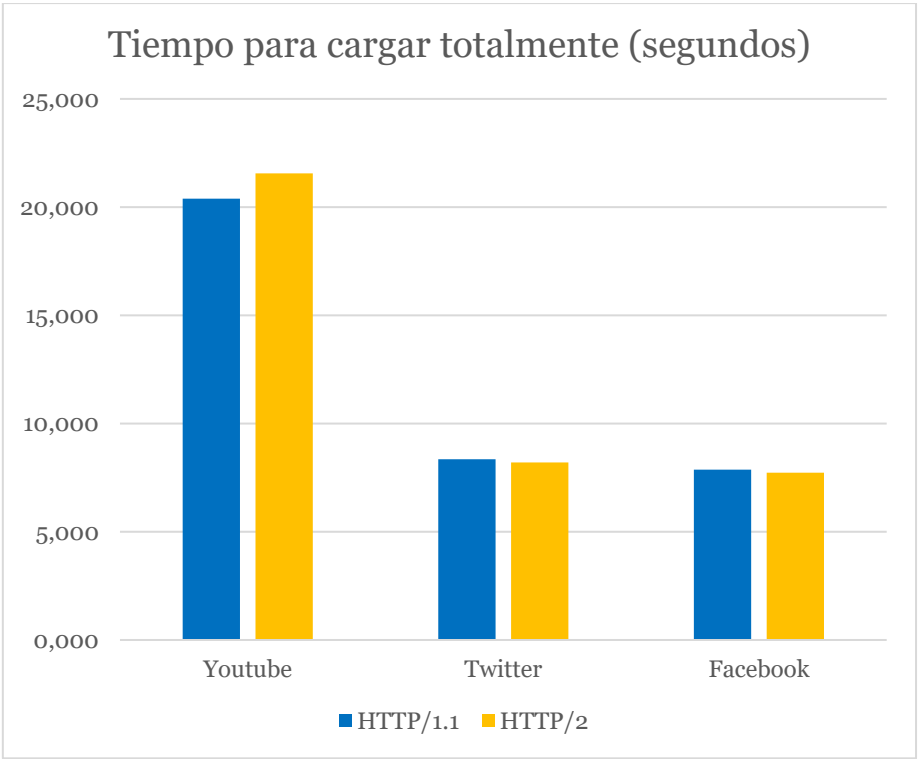
Como se puede observar en la figura anterior el tiempo de carga de las páginas web es significativamente peor en YouTube en HTTP/2, mientras que en Twitter es muy similar y en Facebook es menor en HTTP/2.



En cuanto al tiempo para el primer byte, las 3 páginas tienen un resultado similar.

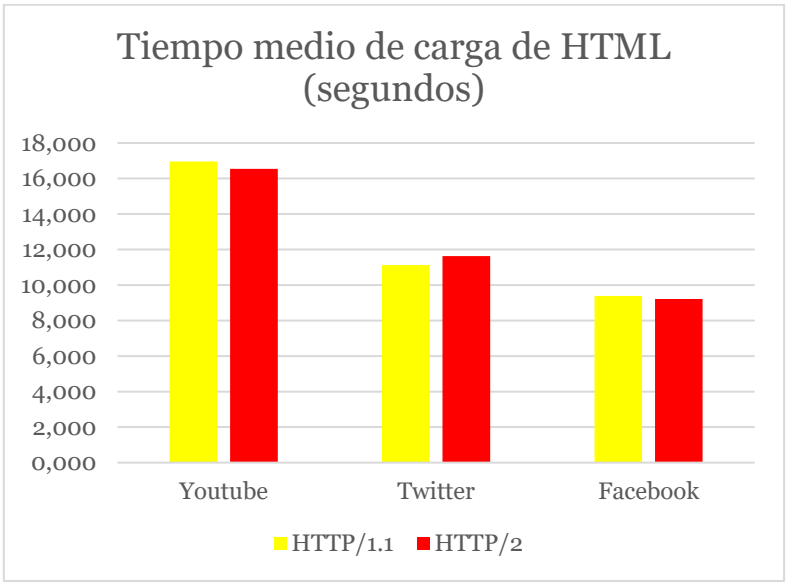


Sin embargo, en el tiempo del comienzo de la renderización de las páginas web es disperso, en HTTP/2 es más rápido en Twitter, pero sin embargo es más lento en YouTube y Facebook. La diferencia de tiempo máximo es de 0.4 segundos.

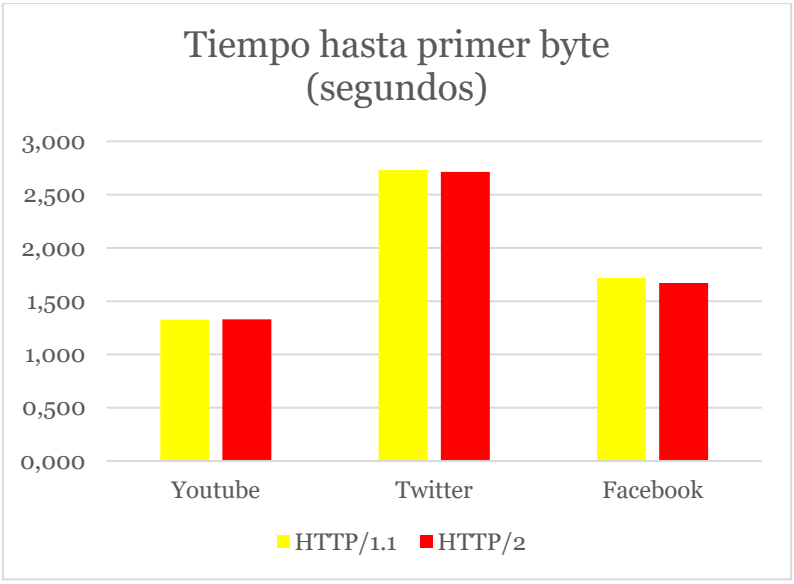


Para concluir con el análisis de 3G con 150 ms de latencia, los tiempos de carga en Twitter y Facebook son más rápidos en HTTP/2 mientras que, en YouTube, el tiempo es peor.

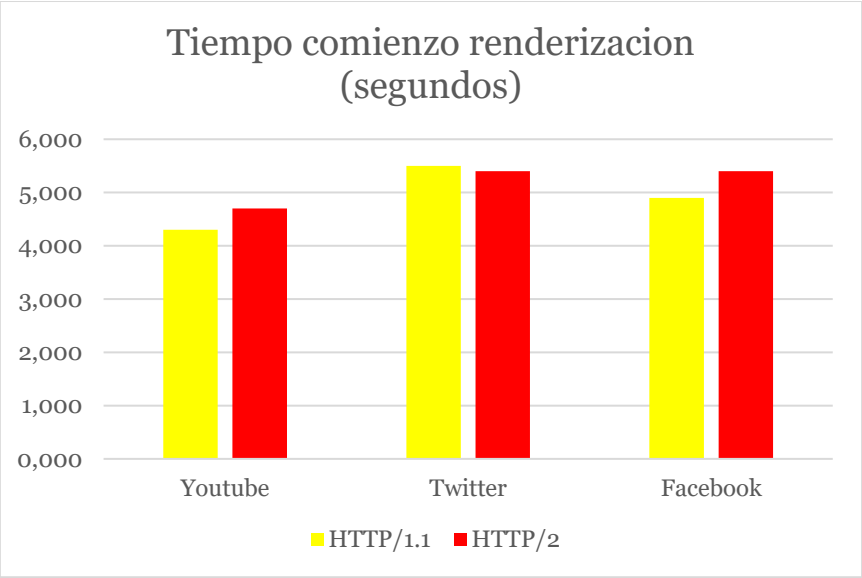
En cuanto al caso intermedio (300 ms) los resultados obtenidos fueron:



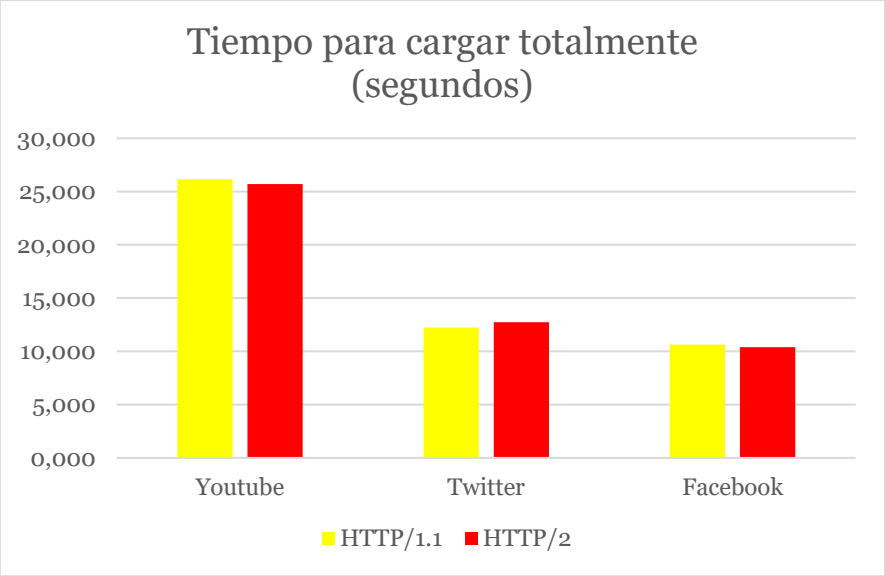
El tiempo de carga es menor en YouTube y Facebook en HTTP/2 mientras que en Twitter es peor. La diferencia es de medio segundo en YouTube y Twitter y de 2 décimas de segundo en Facebook.



En cuanto al tiempo hasta el primer byte, son todos muy parecidos.

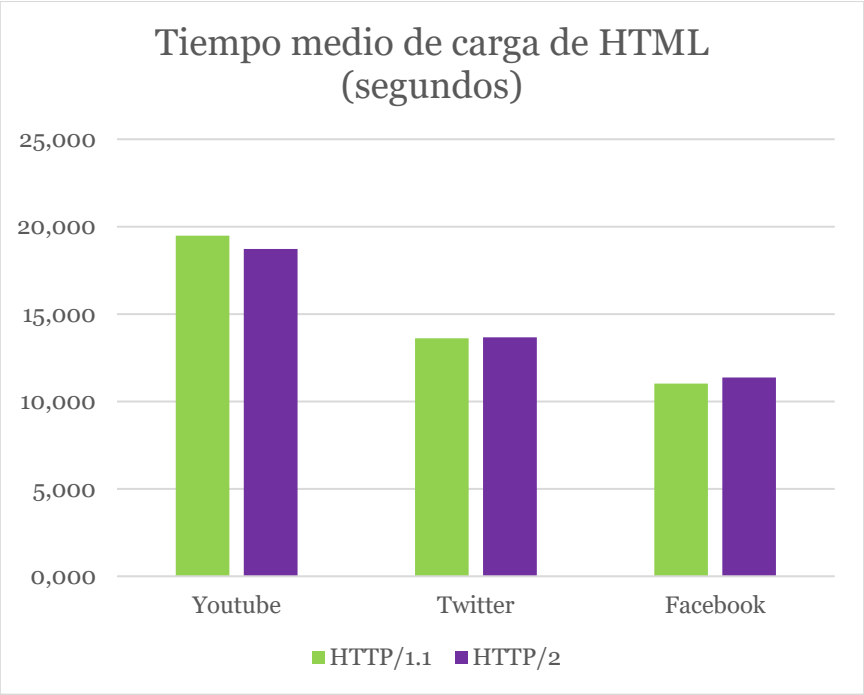


En el tiempo de comienzo de renderización Facebook, mejora respecto al mejor caso y el resto se mantiene en un estado similar con una diferencia máxima de medio segundo entre HTTP/2 y HTTP/1.1.

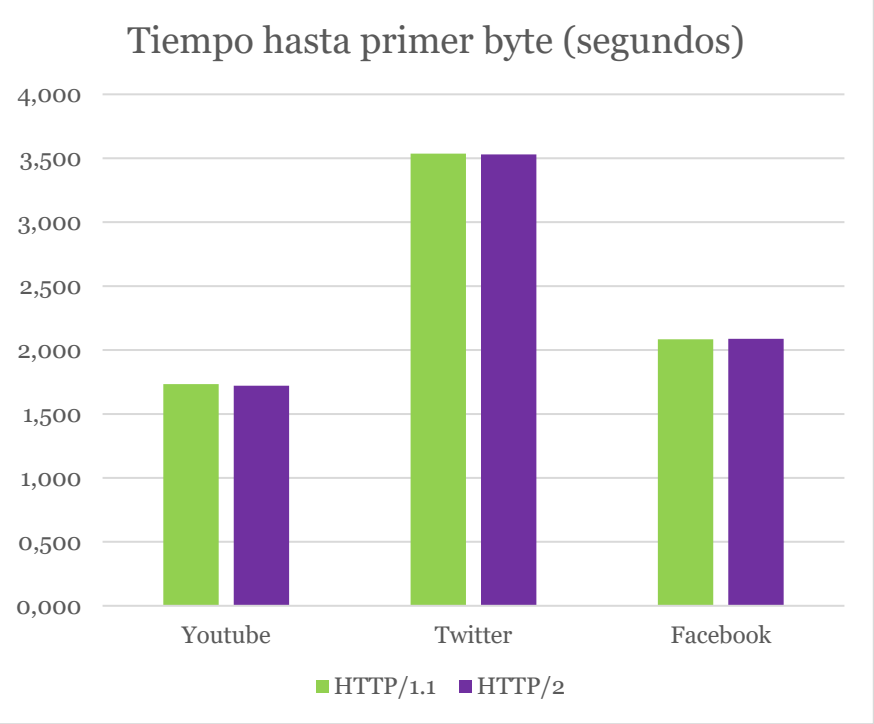


Y para finalizar el caso intermedio, se observa que la diferencia máxima de tiempo para cargar totalmente la página es de 5 décimas de segundo en Twitter.

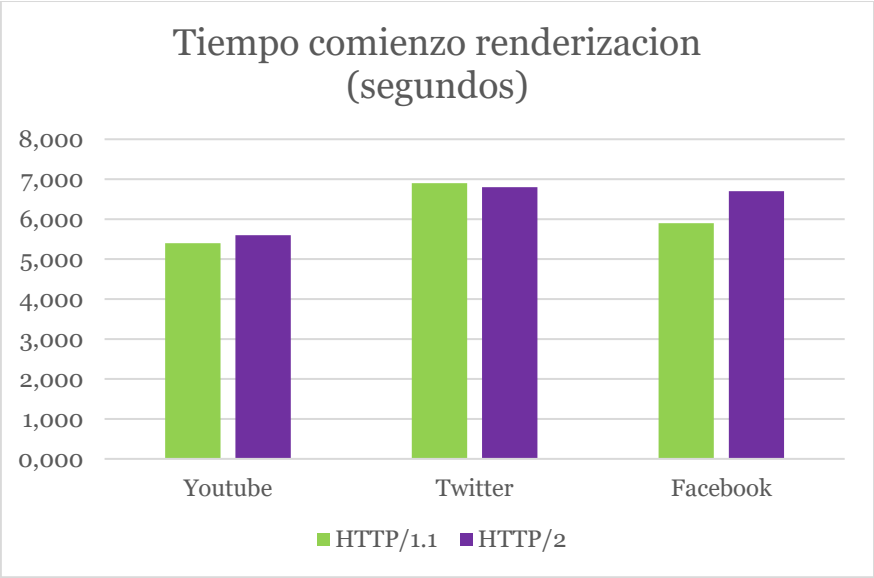
Y por último queda el peor caso (400 ms):



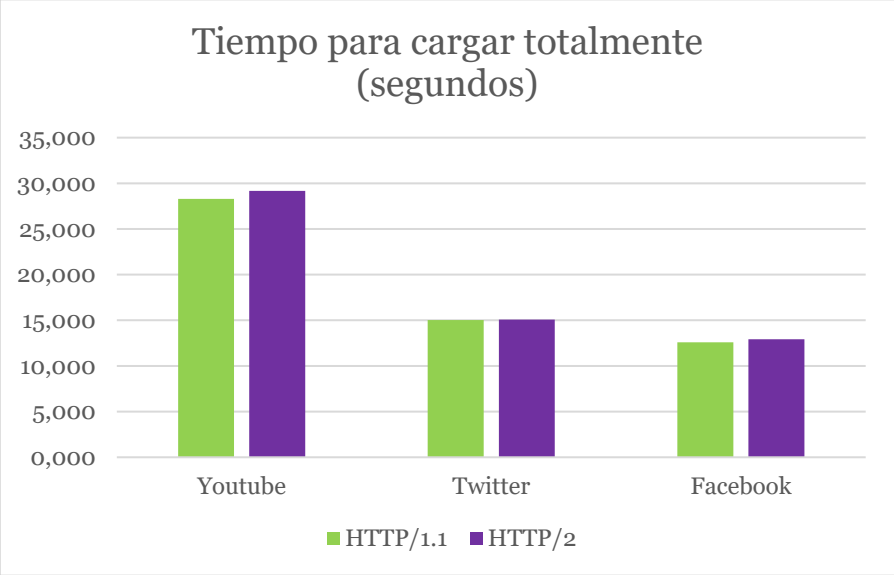
En este caso, en YouTube se mejora sustancialmente el tiempo de carga, en Twitter el tiempo de carga es similar y Facebook empeora respecto a los otros casos.



El tiempo hasta el primer byte es similar en todos.



El tiempo del comienzo de renderización es similar en Twitter, pero Facebook y YouTube empeoran esta estadística. Facebook tarda casi 1 segundo más en HTTP/2 que en HTTP/1.1

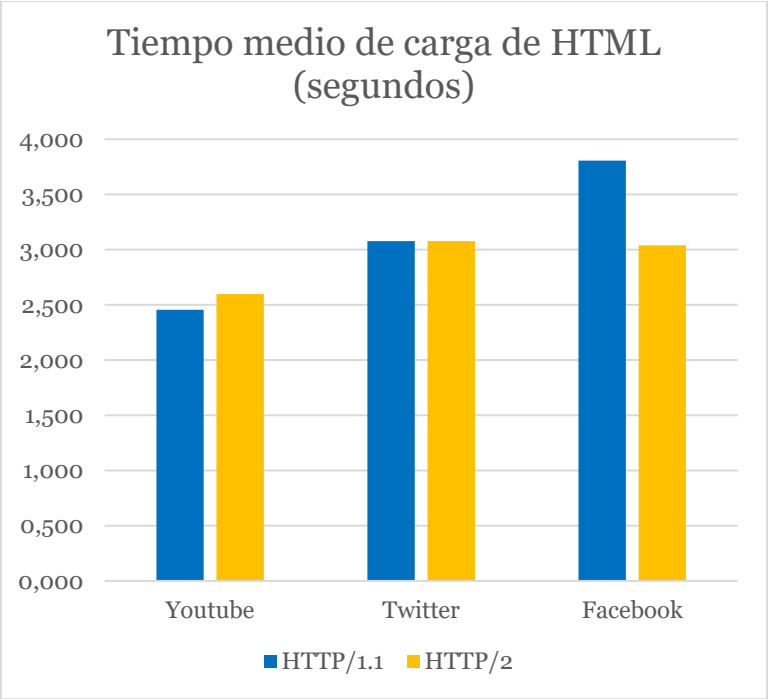


Para finalizar este análisis, el tiempo de carga en YouTube y Facebook es superior en HTTP/2, pero en Twitter es similar el tiempo. El tiempo de YouTube es bastante peor, siendo este de 8 décimas.

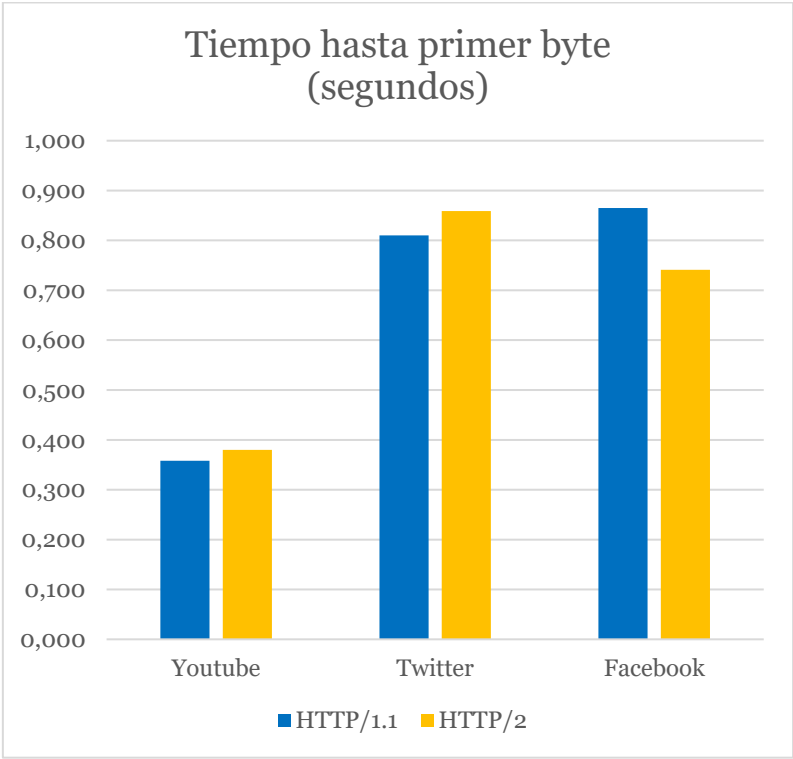
4G

Para realizar este estudio, en la herramienta Webpagetest, se estableció un ancho de banda de 9 Mbps de bajada y 9 Mbps de subida, como mejor caso, se estableció una latencia de 70 ms, como caso medio una latencia de 120 ms y como peor caso una latencia de 170 ms.

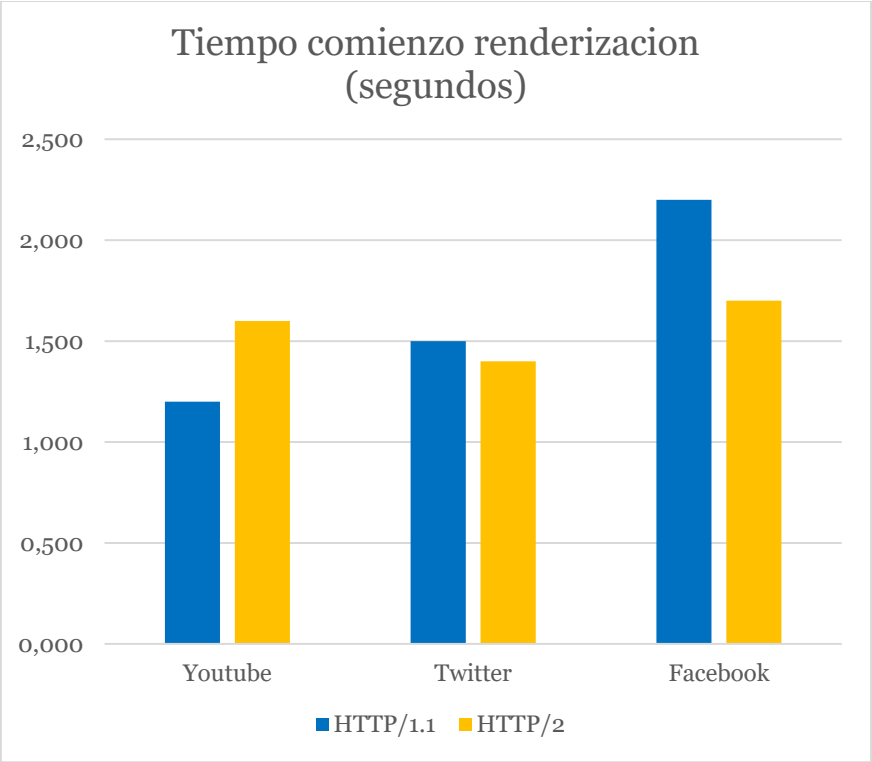
Los resultados obtenidos fueron los siguientes en el mejor caso:



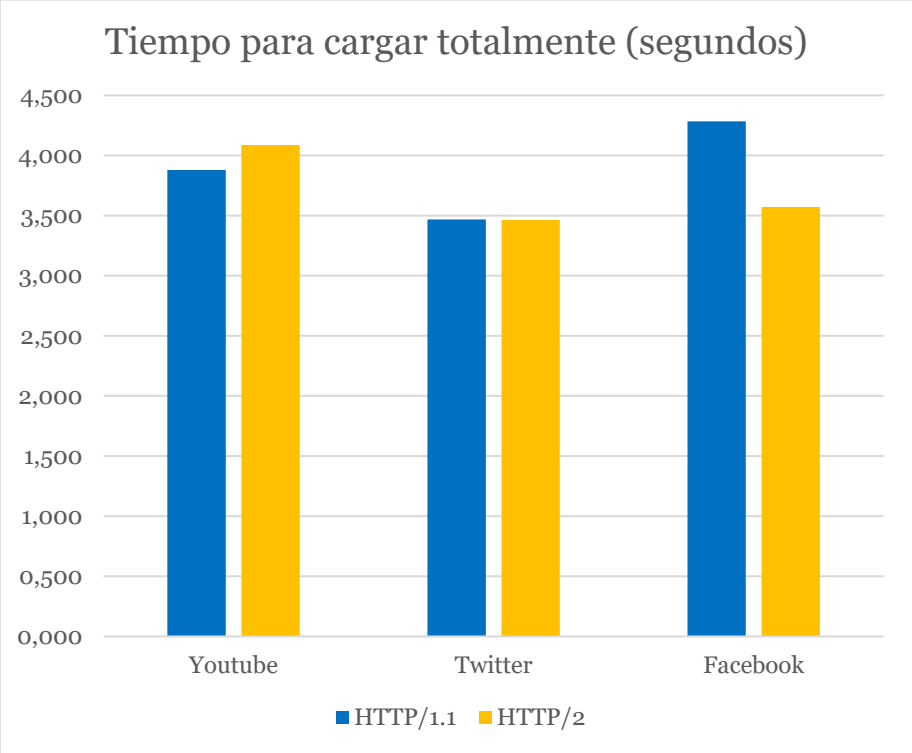
Como se puede observar en la figura anterior el tiempo de carga de las páginas web es mayor en YouTube en HTTP/2, mientras que en Twitter es muy similar y en Facebook es significativamente menor en HTTP/2.



En cuanto al tiempo para el primer byte, las 3 páginas tienen un resultado similar.

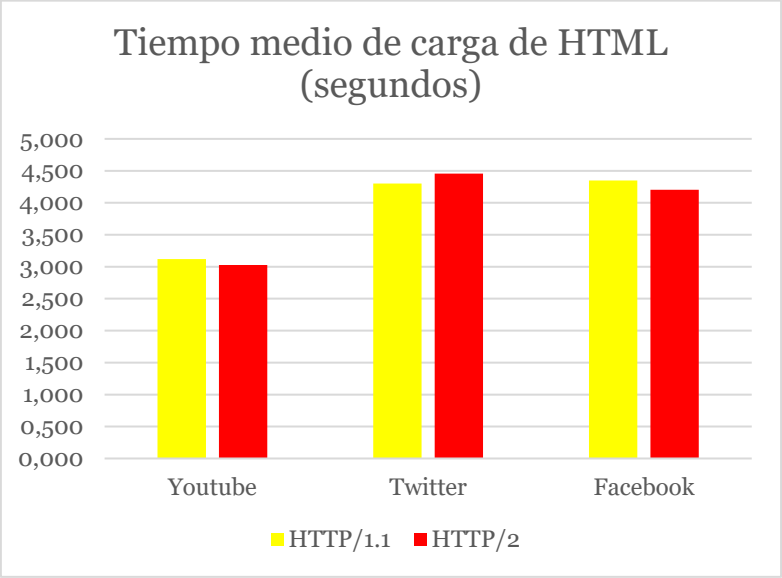


Sin embargo, en el tiempo del comienzo de la renderización de las páginas web es menor en Facebook y Twitter que en YouTube.

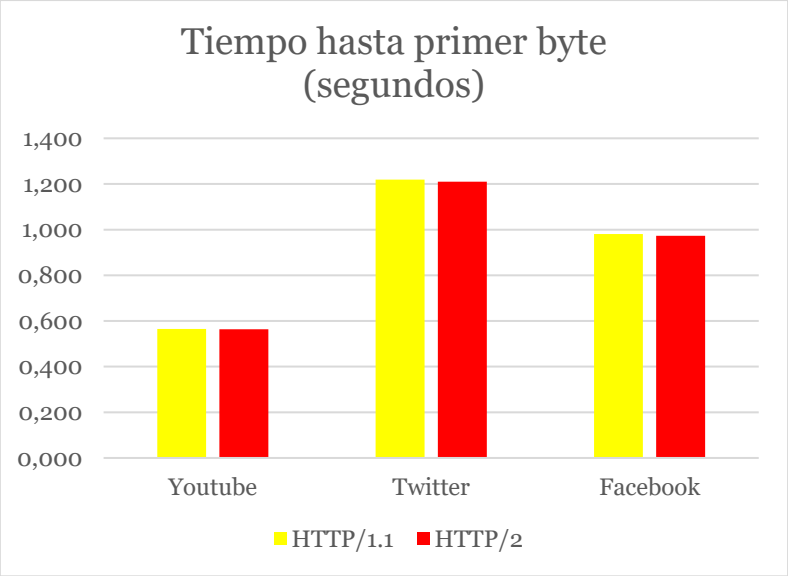


Para concluir con el análisis de 4G con 70 ms de latencia, se observa que el tiempo es mucho menor en Facebook en HTTP/2, mientras que en Twitter es bastante similar y en YouTube empeora por unas décimas de segundo.

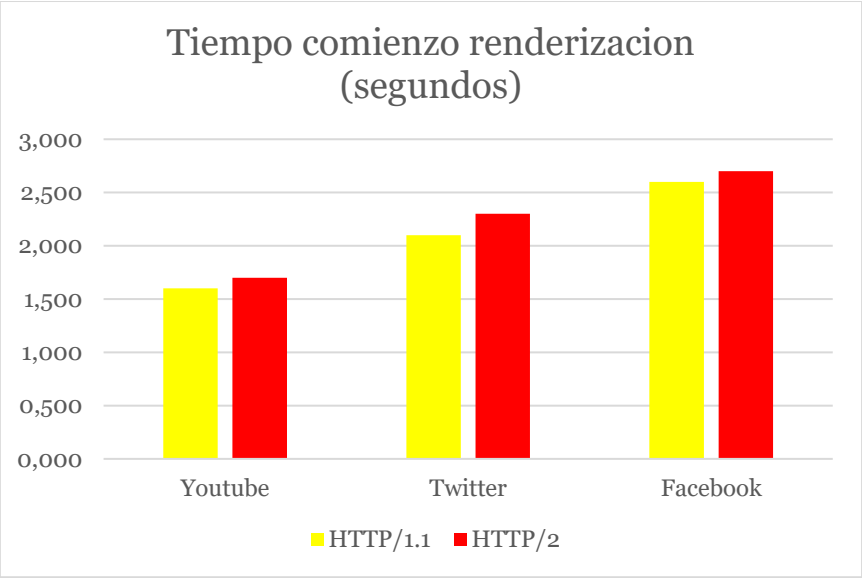
En cuanto al caso intermedio los resultados obtenidos fueron:



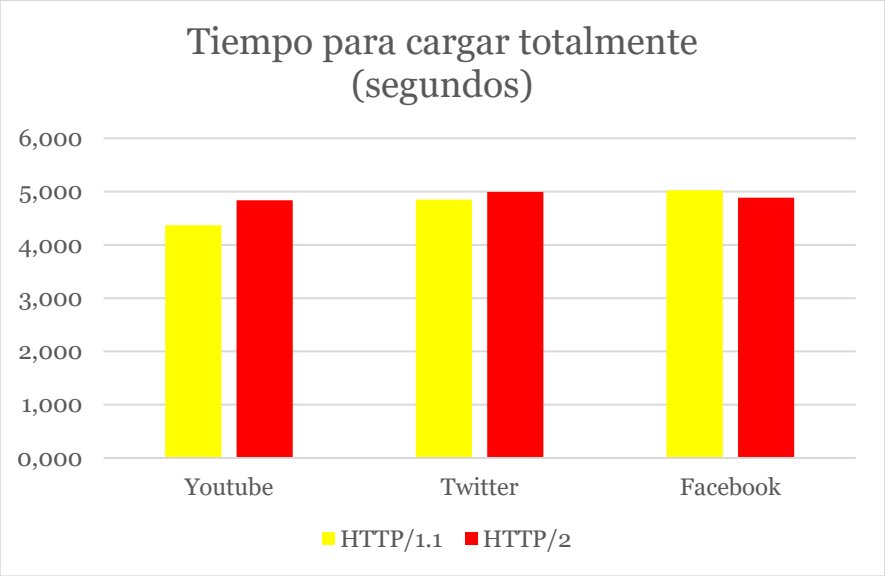
El tiempo de carga es un poco mejor en Facebook y YouTube, pero Twitter tarda más en cargar la página.



En cuanto al tiempo hasta el primer byte, son todos muy parecidos.

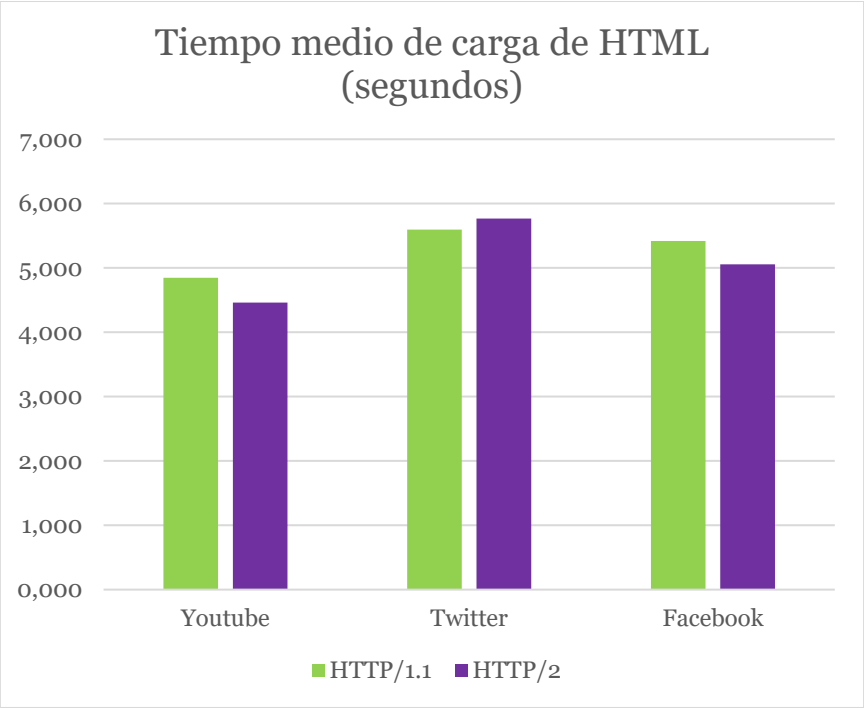


En el tiempo de comienzo de renderización, todos son peores en HTTP/2 que en HTTP/1.1.

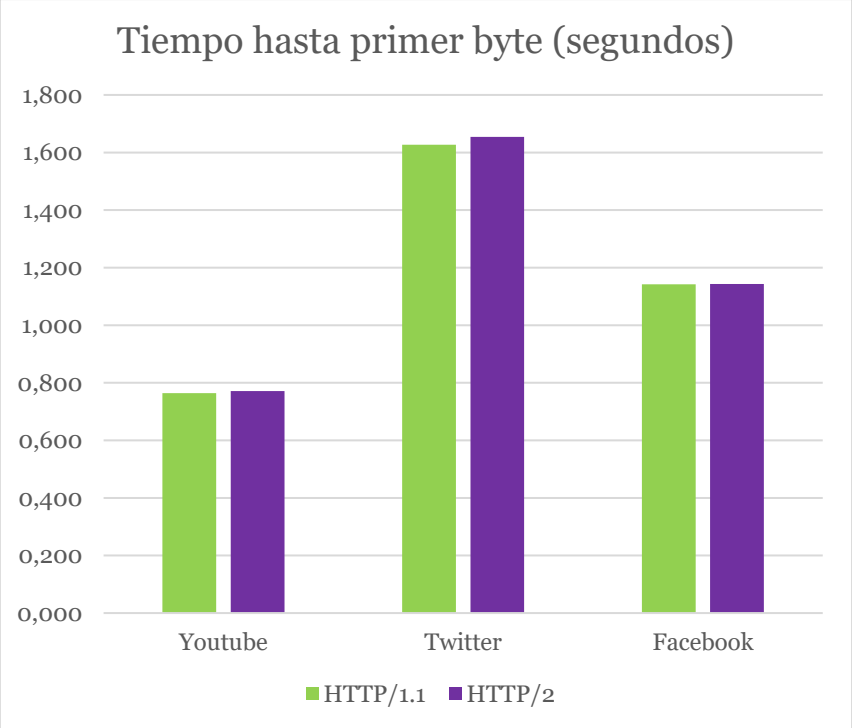


Y para finalizar el caso intermedio, se observa que los tiempos son peores en YouTube y Twitter, mientras que en Facebook el tiempo es mejor.

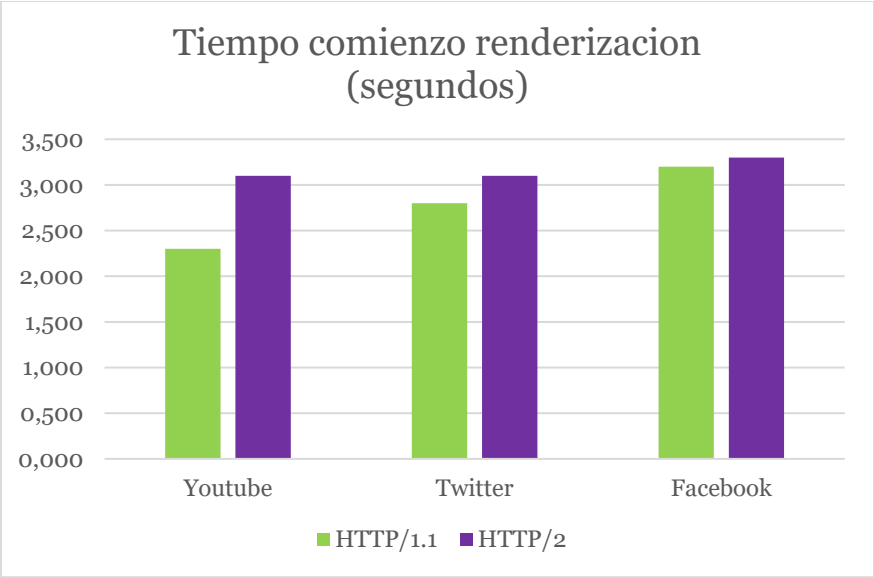
Y por último queda el peor caso:



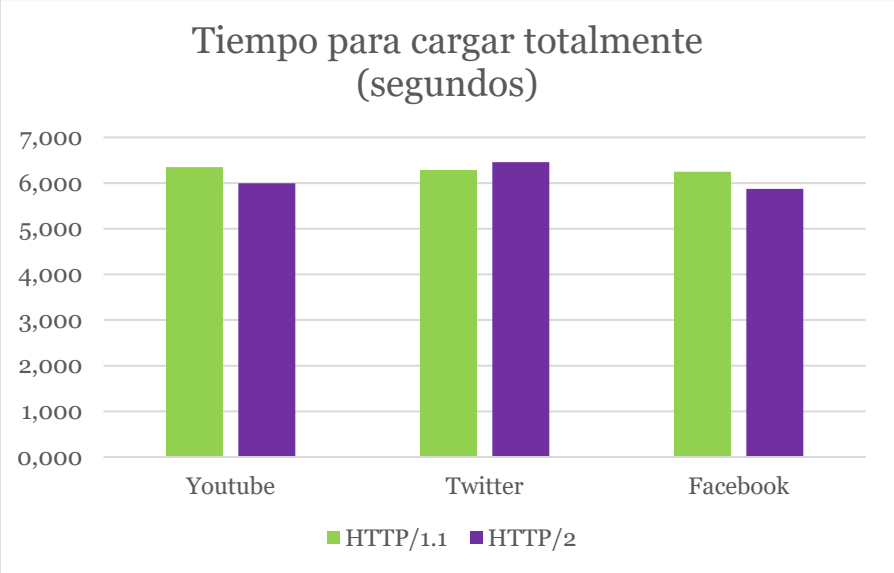
En este caso, en YouTube y Facebook, el tiempo de carga es menor, mientras que Twitter es un poco mayor en HTTP/2.



El tiempo hasta el primer byte es similar en todos.



El tiempo del comienzo de renderización es peor en HTTP/2 en todos.

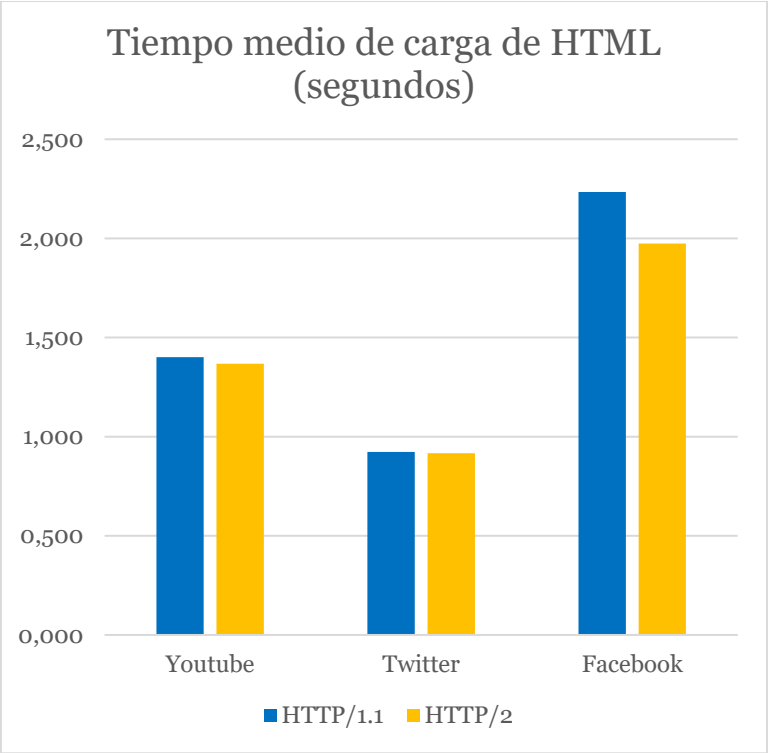


Para concluir este análisis se observa que YouTube y Facebook cargan completamente la página más rápido en HTTP/2 y Twitter empeora con esta versión del protocolo.

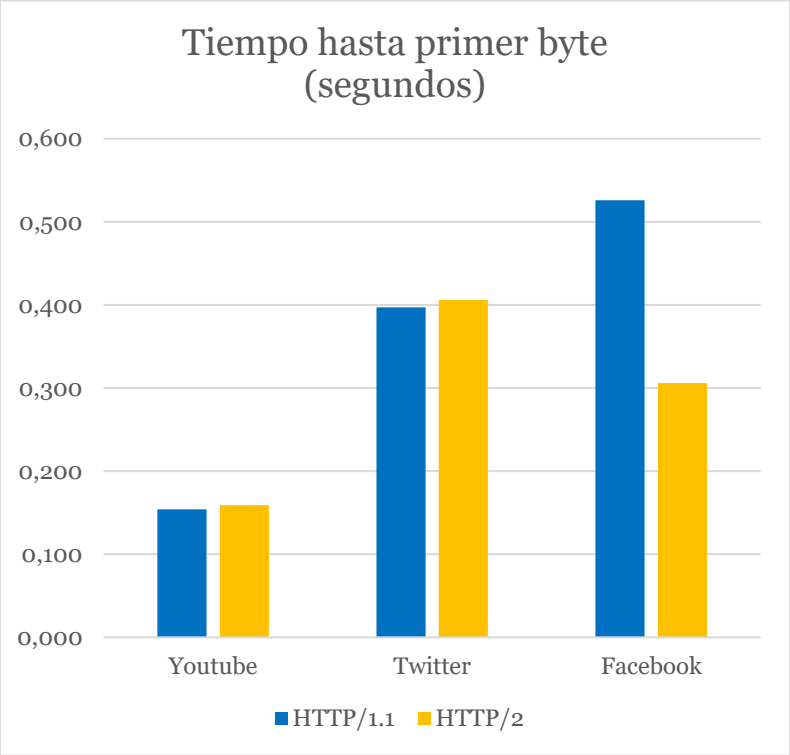
Fibra óptica

Para realizar este estudio, en la herramienta Webpagetest, se estableció un ancho de banda de 300 Mbps de bajada y 300 Mbps de subida, como mejor caso, se estableció una latencia de 20 ms, como caso medio una latencia de 50 ms y como peor caso una latencia de 80 ms.

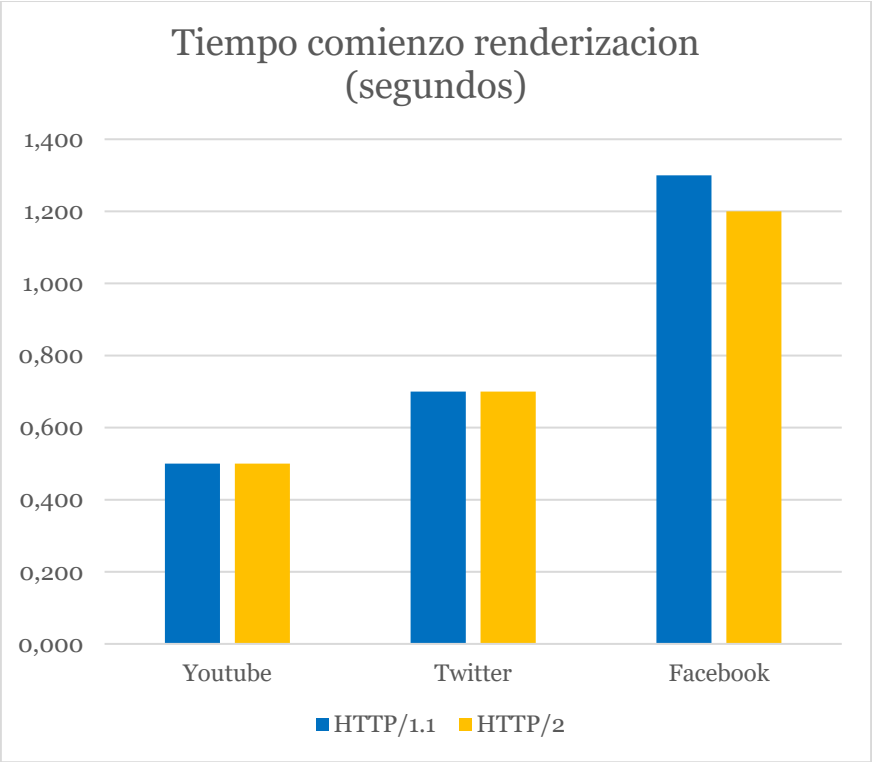
Los resultados obtenidos fueron los siguientes en el mejor caso:



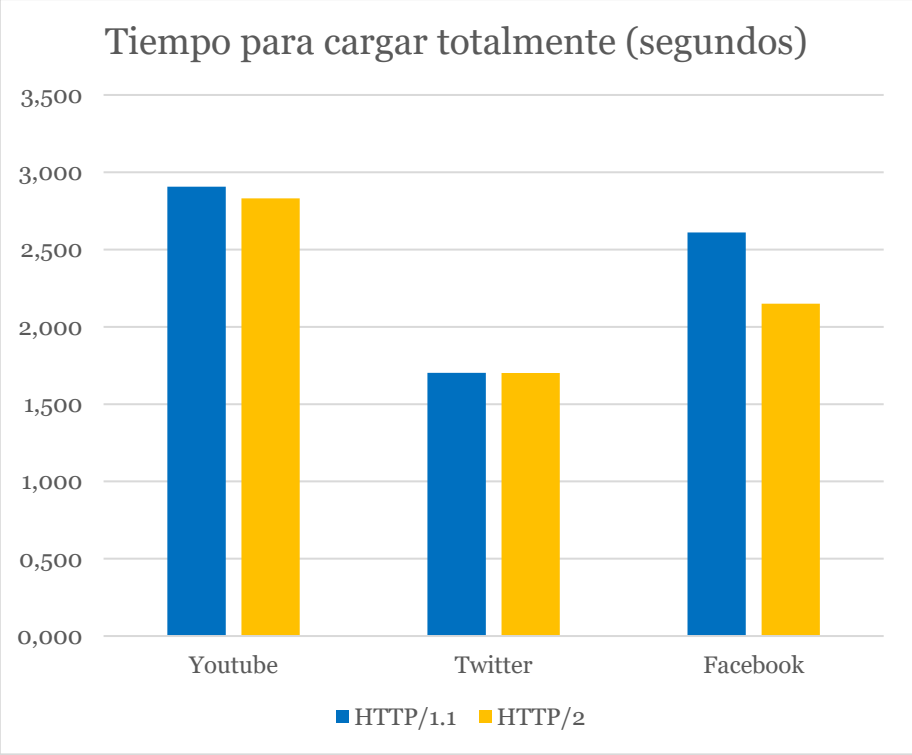
Como se puede observar en la figura anterior el tiempo de carga de las páginas web es mejor en HTTP/2. La mayor diferencia se aprecia en Facebook, cuya diferencia es de 3 décimas de segundo.



En cuanto al tiempo para el primer byte, YouTube y Twitter tienen tiempos similares, pero Facebook mejora sustancialmente este aspecto.

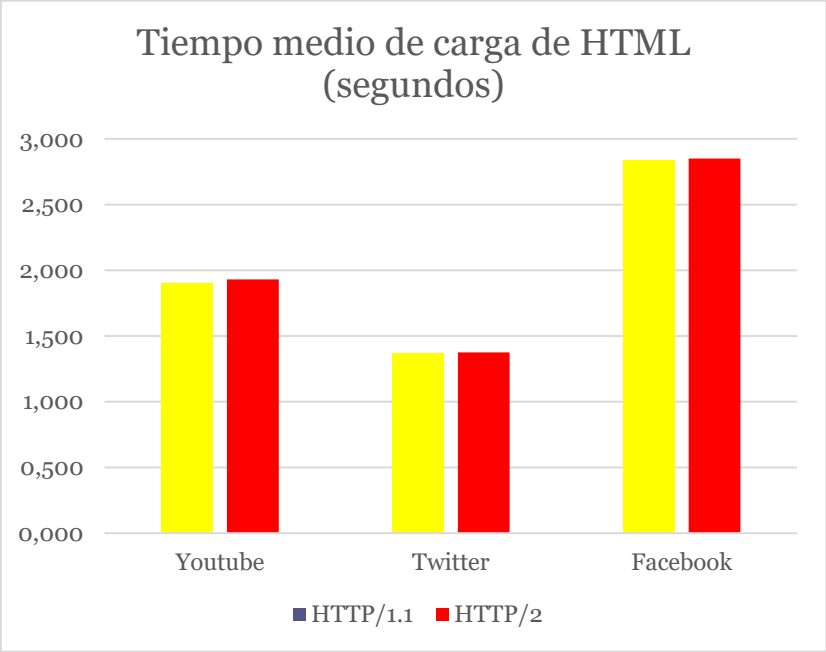


Sin embargo, en el tiempo del comienzo de la renderización de las páginas web, es similar en todas, pero Facebook tarda 1 décima de segundo más.

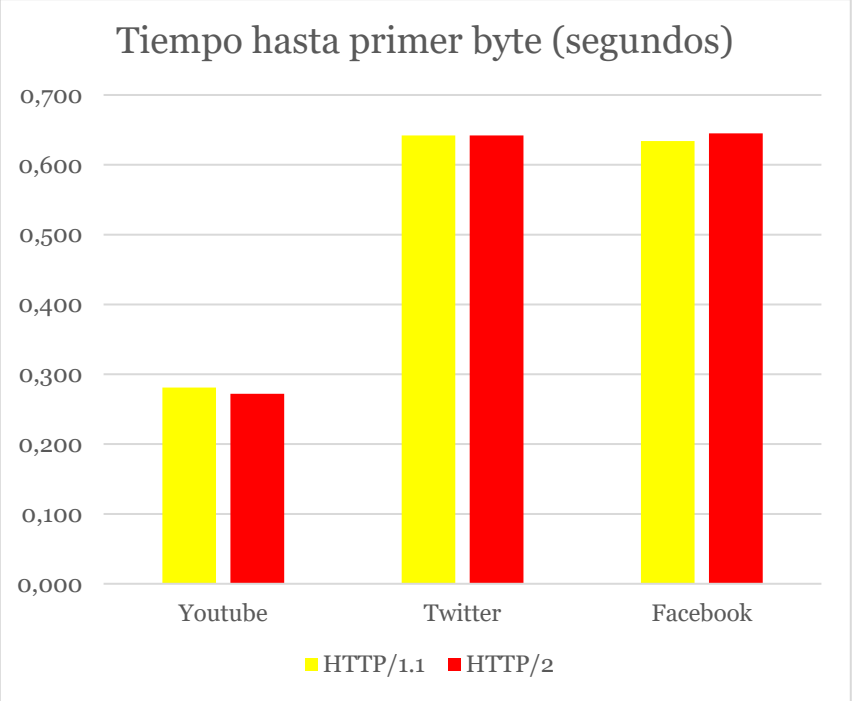


Para concluir con el análisis de Fibra Óptica con 20 ms de latencia, se aprecia cómo se experimenta una mejora en las tres páginas web.

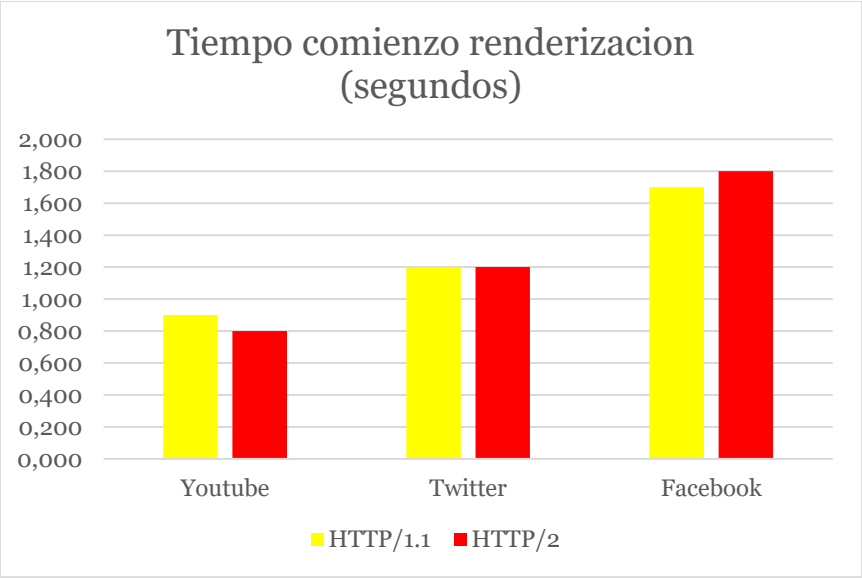
En cuanto al caso intermedio los resultados obtenidos fueron:



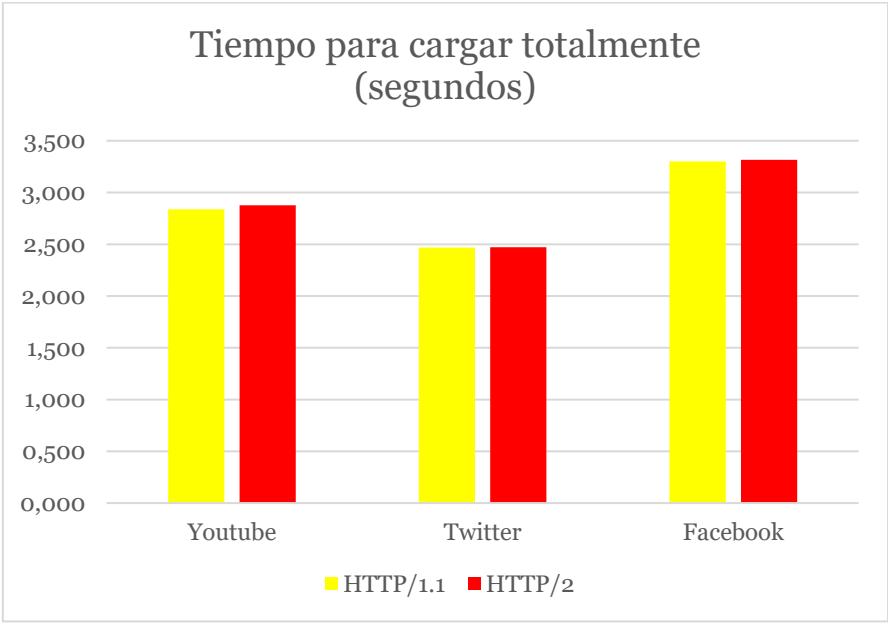
El tiempo de carga es bastante parecido en todas las páginas.



En cuanto al tiempo hasta el primer byte, son todos muy parecidos.

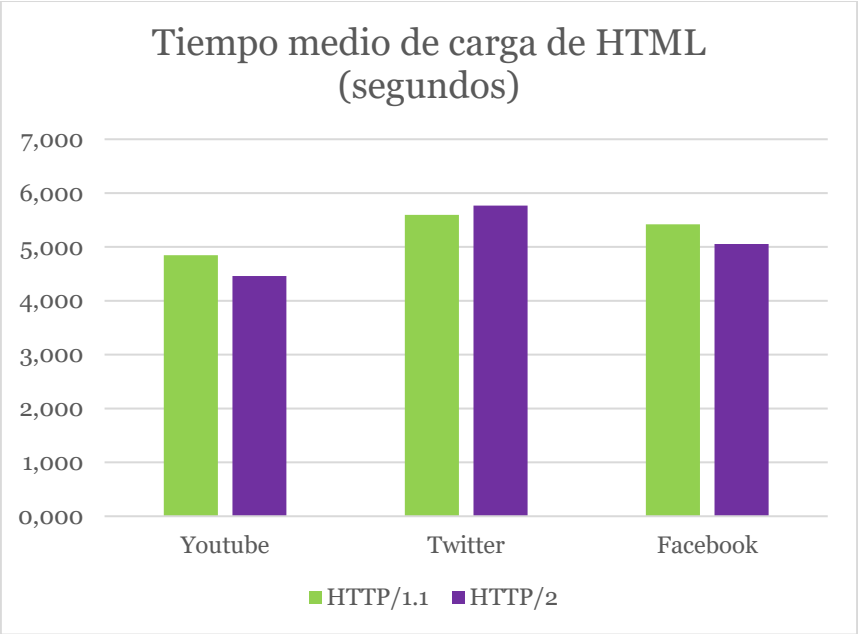


En el tiempo de comienzo de renderización, el tiempo es similar en todos.

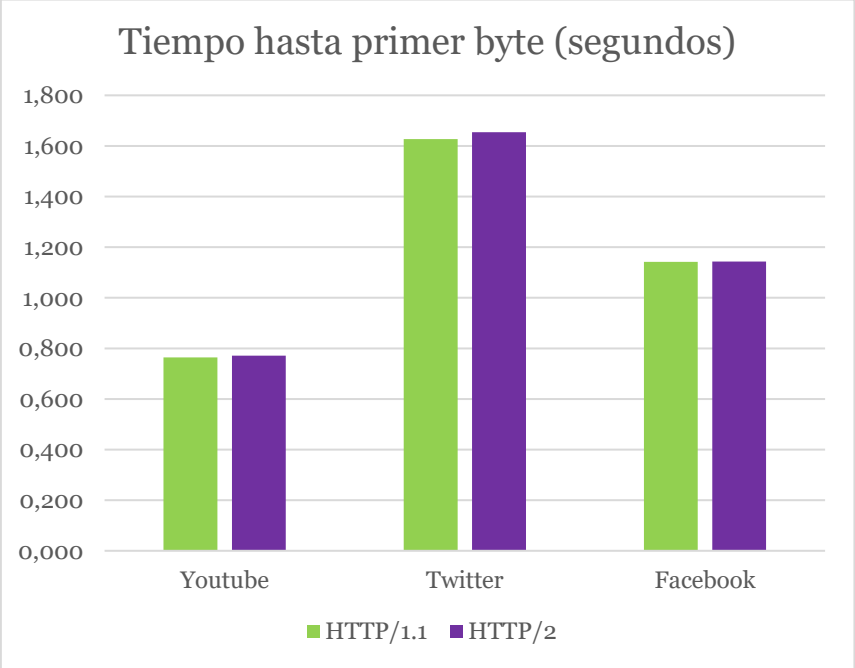


Y para finalizar el caso intermedio, se observa que los tiempos son muy similares.

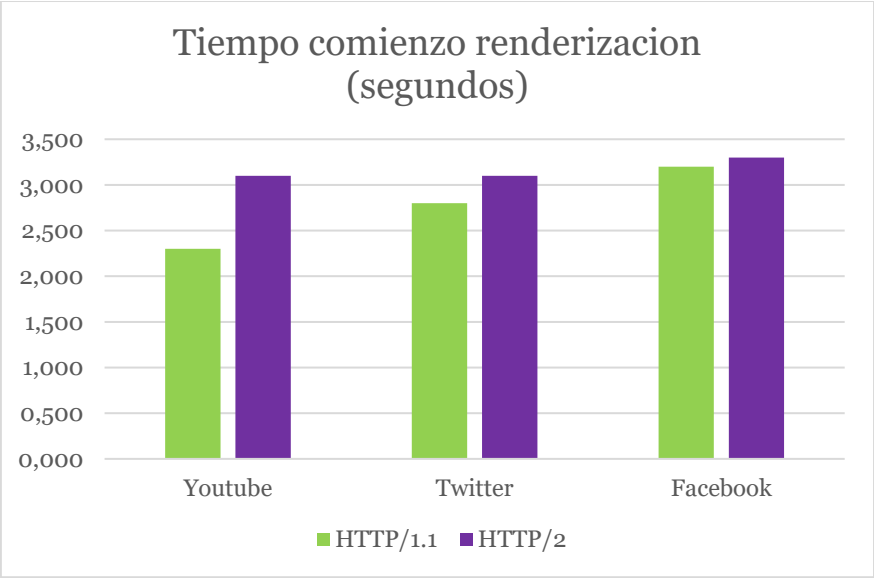
Y por último queda el peor caso:



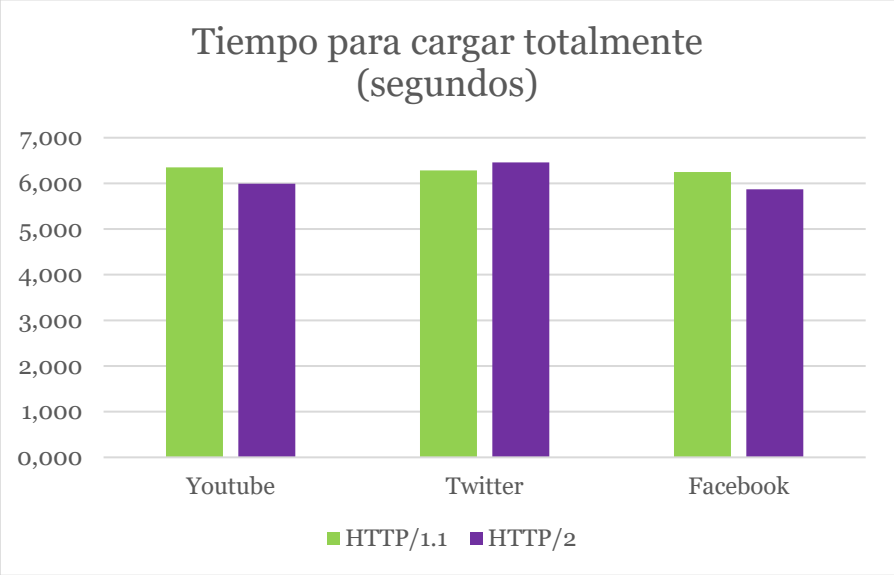
En este caso, en YouTube y Facebook, el tiempo es mejor en HTTP/2.



El tiempo hasta el primer byte es similar en todos.



El tiempo del comienzo de renderización es similar en todos, pero peor en HTTP/2 en todos.



Para finalizar este análisis, YouTube y Facebook mejoran el tiempo de carga en HTTP/2, pero Twitter lo empeora en este nuevo protocolo.

El primer ejemplo tratado es bajo las circunstancias de una red que trabaja con ADSL, elaborado con la herramienta Webpagetest, es importante remarcar que a medida que la latencia aumenta el tiempo medio de carga de la página disminuye considerablemente con HTTP/2, pero sin embargo el tiempo para cargar por completo la página, YouTube empeora con este protocolo.

El segundo ejemplo estudiado es con las circunstancias de una red 3G, elaborado con la herramienta Webpagetest, en este caso tanto el tiempo de carga como el tiempo

para cargar por completo la página son muy similares o peores en HTTP/2 bajo todas circunstancias.

El tercer ejemplo es sobre una red 4G, elaborado con la herramienta Webpagetest, en este caso vemos que tanto el tiempo de carga como el tiempo para cargar totalmente la página son mejores en Facebook bajo HTTP/2, en YouTube es mejor solo en el peor caso y Twitter tiene un rendimiento similar, pero empeora con HTTP/2.

El cuarto ejemplo trata sobre una red de fibra óptica, realizado con la herramienta Webpagetest y podemos observar como casi todos los tiempos de carga son mejores en HTTP/2 en Facebook y YouTube, Twitter tiene un tiempo muy similar.

5.2.2 Pruebas sobre una red física real

Para realizar este estudio se empleó una conexión real de 300 Mbps de descarga y de subida.

#	VELOCIDAD BAJADA	VELOCIDAD SUBIDA	SERVIDOR	LATENCIA
1	298.51	312.86	Movistar	25

Figura 11 Test de velocidad

La latencia a las 3 páginas web se realizó mediante el comando “ping”. Las latencias obtenidas fueron 9 ms para Facebook y YouTube y 46 para Twitter, como se aprecia en las siguientes imágenes:

```
C:\Users\javier>ping www.facebook.com

Haciendo ping a star-z-mini.c10r.facebook.com [31.13.83.38] con 32 bytes de datos:
Respuesta desde 31.13.83.38: bytes=32 tiempo=9ms TTL=55
Respuesta desde 31.13.83.38: bytes=32 tiempo=9ms TTL=55
Respuesta desde 31.13.83.38: bytes=32 tiempo=9ms TTL=55
Respuesta desde 31.13.83.38: bytes=32 tiempo=9ms TTL=55

Estadísticas de ping para 31.13.83.38:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 9ms, Máximo = 9ms, Media = 9ms
```

Figura 12 Latencia Facebook

```
C:\Users\javier>ping www.twitter.com

Haciendo ping a twitter.com [104.244.42.129] con 32 bytes de datos:
Respuesta desde 104.244.42.129: bytes=32 tiempo=46ms TTL=54
Respuesta desde 104.244.42.129: bytes=32 tiempo=46ms TTL=54
Respuesta desde 104.244.42.129: bytes=32 tiempo=46ms TTL=54
Respuesta desde 104.244.42.129: bytes=32 tiempo=46ms TTL=54

Estadísticas de ping para 104.244.42.129:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 46ms, Máximo = 46ms, Media = 46ms
```

Figura 13 Latencia Twitter

```
C:\Users\javier>ping www.youtube.com

Haciendo ping a youtube-ui.l.google.com [172.217.17.14] con 32 bytes de datos:
Respuesta desde 172.217.17.14: bytes=32 tiempo=9ms TTL=53
Respuesta desde 172.217.17.14: bytes=32 tiempo=9ms TTL=53
Respuesta desde 172.217.17.14: bytes=32 tiempo=9ms TTL=53
Respuesta desde 172.217.17.14: bytes=32 tiempo=9ms TTL=53

Estadísticas de ping para 172.217.17.14:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 9ms, Máximo = 9ms, Media = 9ms

C:\Users\javier>_
```

Figura 14 Latencia YouTube

Después de obtener la latencia, se realizó el siguiente estudio:

Primero, se vació la cache de Chrome, para acceder a este apartado, se puede acceder mediante la siguiente pestaña:

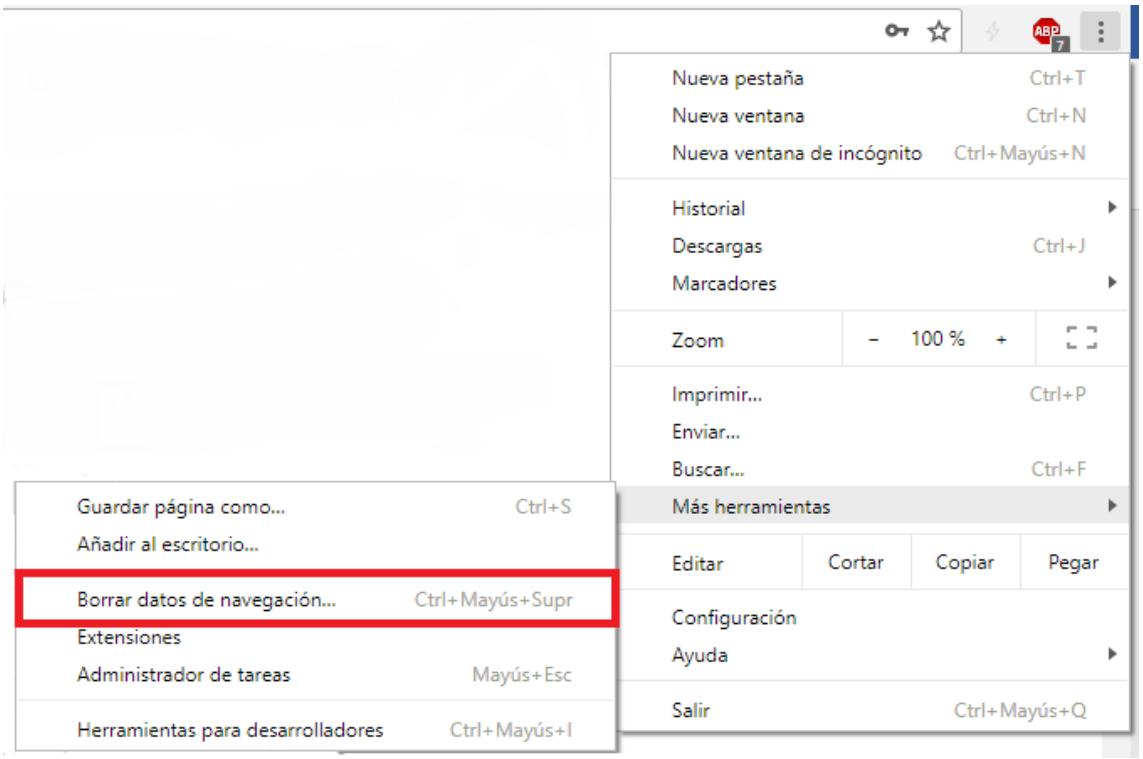


Figura 15 Guía para borrar datos

O mediante el atajo, Control+Mayúsculas+Suprimir. Se procedió a borrar todos los datos de navegación y se procedió con el siguiente paso.

Una vez realizado, se abrió un terminal de símbolo de sistema y se cambió el directorio al lugar donde se tiene instalado el navegador. Puesto que Chrome soporta argumentos al iniciar el navegador, se invocó el navegador con el argumento –disable-http2, que desactiva http2 en la pestaña elegida.

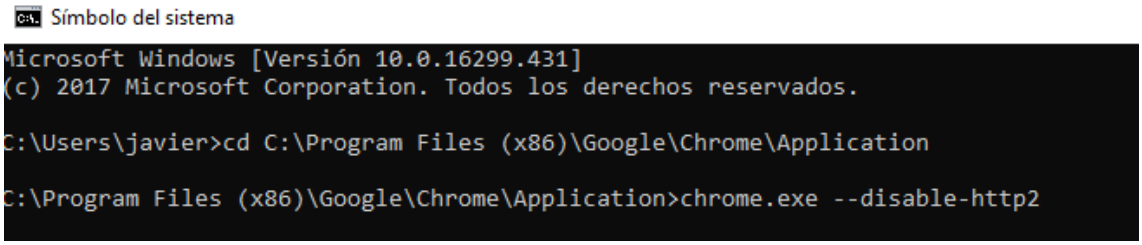


Figura 16 Deshabilitar HTTP/2 en Chrome

A continuación, se seleccionó la opción de herramientas para desarrolladores que puede ser invocada mediante el atajo Control+Mayúsculas+I o mediante la siguiente pestaña:

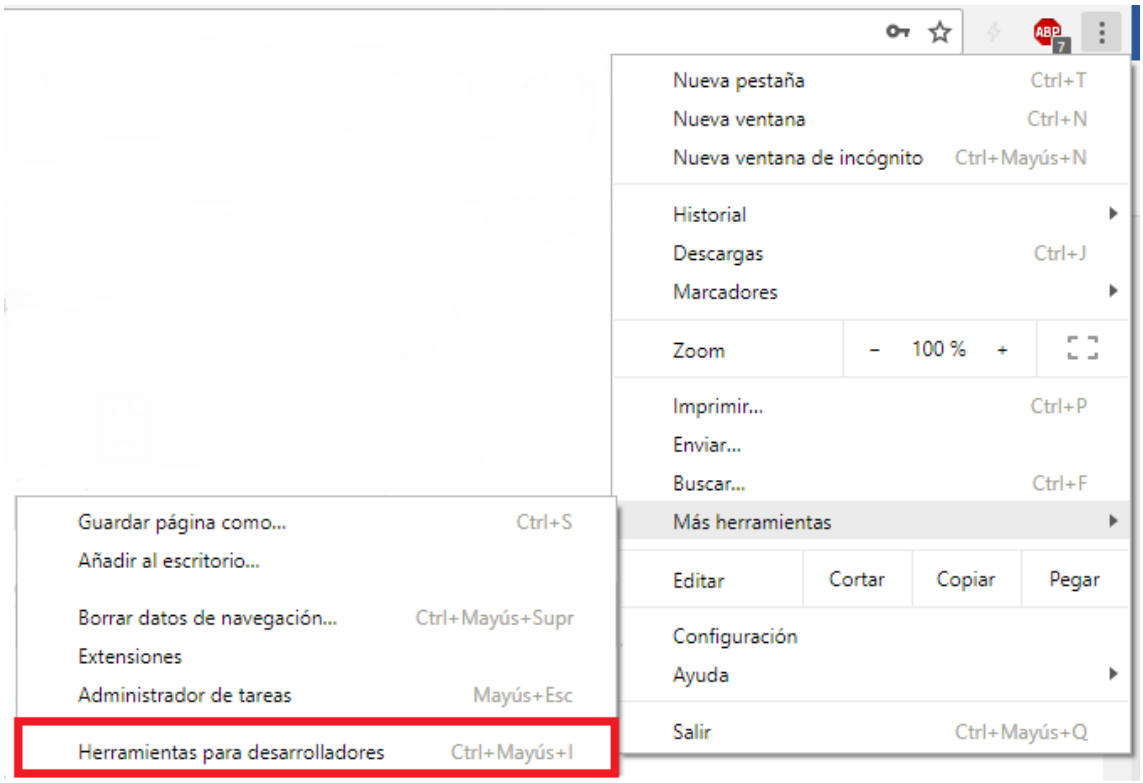


Figura 17 Guía para Herramienta de Desarrolladores

El objetivo de esta herramienta es visualizar el tiempo tardado por la página web en cargarse, así como el protocolo que se ha utilizado. Una vez se cargó la herramienta, se cargaron las 3 páginas web en HTTP/1.1.

YouTube fue la primera página cargada y los resultados fueron los siguientes:

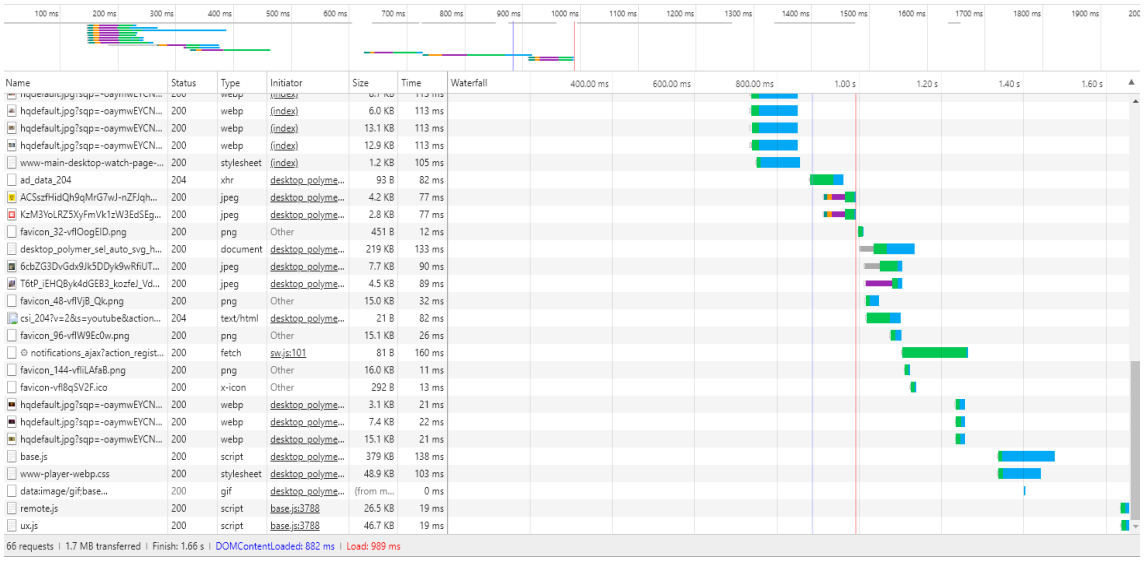


Figura 18 YouTube HTTP/1.1

DOMContentLoaded: 882 ms | Load: 989 ms

El tiempo de carga medio fue de 882 ms mientras que el tiempo total para cargar la página al completo fue de 989 ms.

Finalmente, para comprobar que esta página estaba usando http/1.1 se abrió un recurso para ver la estructura de la petición y la respuesta:

Headers Preview Response Timing

General
Request URL: https://fonts.googleapis.com/css?family=YT%20Sans%3A300%2C500%2C700
Request Method: GET
Status Code: 200 OK
Remote Address: 216.58.201.138:443
Referrer Policy: no-referrer-when-downgrade

Response Headers view source
Access-Control-Allow-Origin: *
Alt-Svc: quic=":443"; ma=2592000; v="43,42,41,39,35"
Cache-Control: private, max-age=86400, stale-while-revalidate=604800
Content-Encoding: br
Content-Type: text/css; charset=utf-8
Date: Sun, 10 Jun 2018 22:21:07 GMT
Expires: Sun, 10 Jun 2018 22:21:07 GMT
Last-Modified: Sun, 10 Jun 2018 22:21:07 GMT
Link: <https://fonts.gstatic.com>; rel=preconnect; crossorigin
Server: ESF
Strict-Transport-Security: max-age=600
Timing-Allow-Origin: *
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block

Request Headers view source
Accept: text/css,*/*;q=0.1
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9
Connection: keep-alive
Host: fonts.googleapis.com
Referer: https://www.youtube.com/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36
X-Client-Data: CJC2yQEIpLbJAQjBtskBCKmdygEI153KAQioo8oB

Figura 19 Comprobación YouTube HTTP/1.1

La siguiente página cargada fue Twitter, con los siguientes resultados:

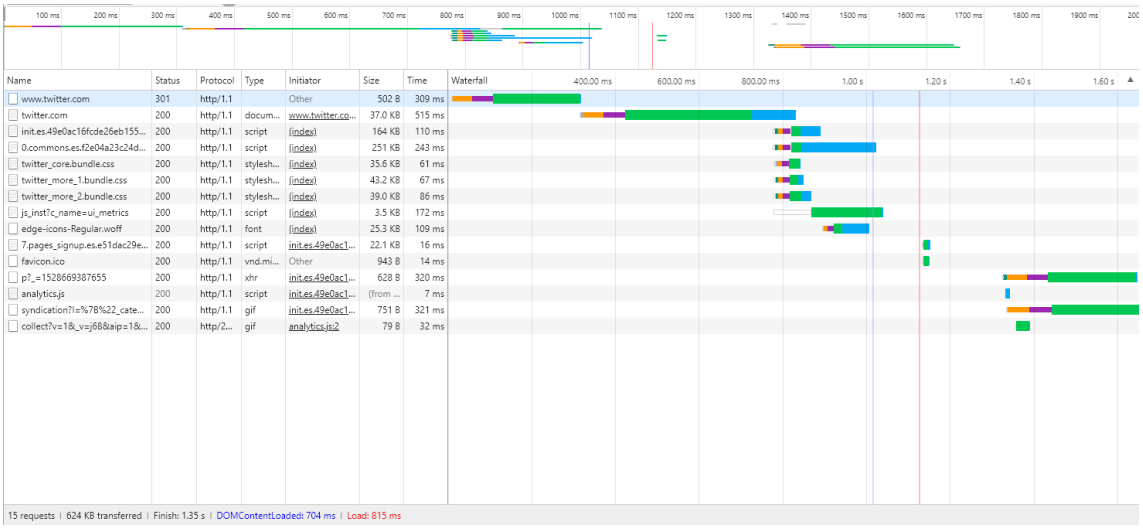


Figura 20 Twitter HTTP/1.1

DOMContentLoaded: 704 ms | Load: 815 ms

El tiempo de carga medio fue de 704 ms mientras que el tiempo total para cargar la página al completo fue de 815 ms.

Finalmente, para comprobar que esta página estaba usando http/1.1 se abrió un recurso para ver la estructura de la petición y la respuesta:

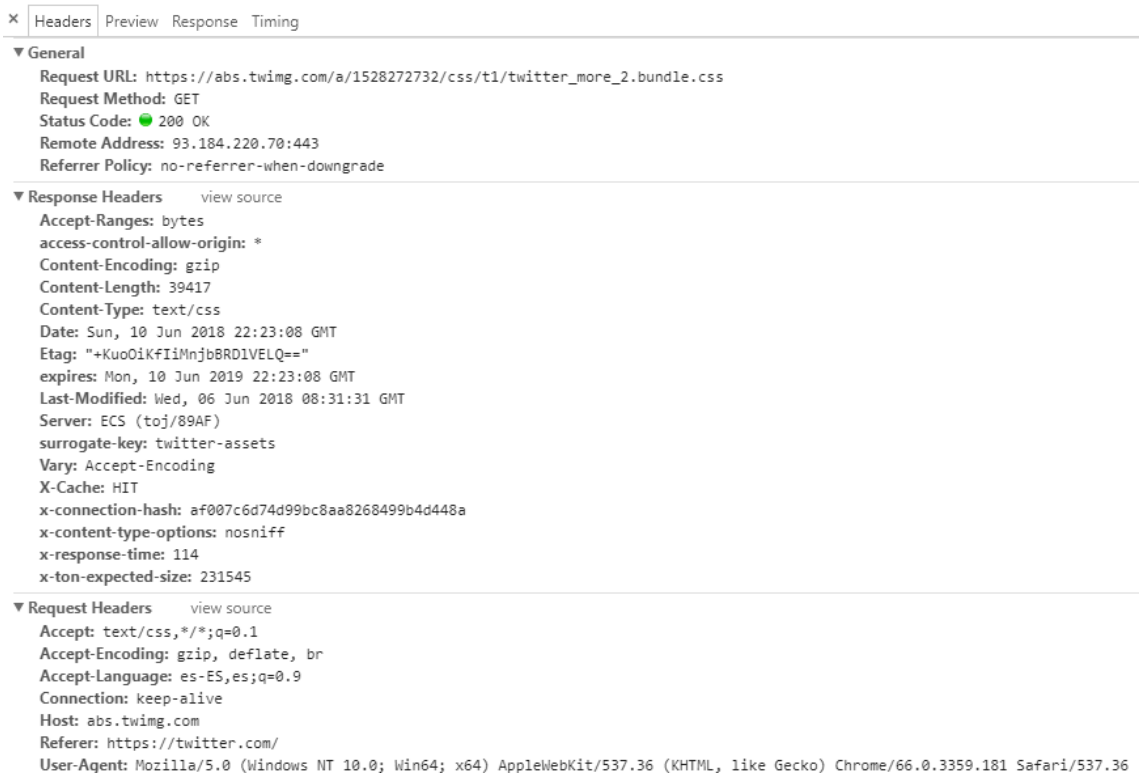


Figura 21 Comprobación Twitter HTTP/1.1

Facebook fue la última página probada en HTTP/1.1 y sus resultados fueron los siguientes:

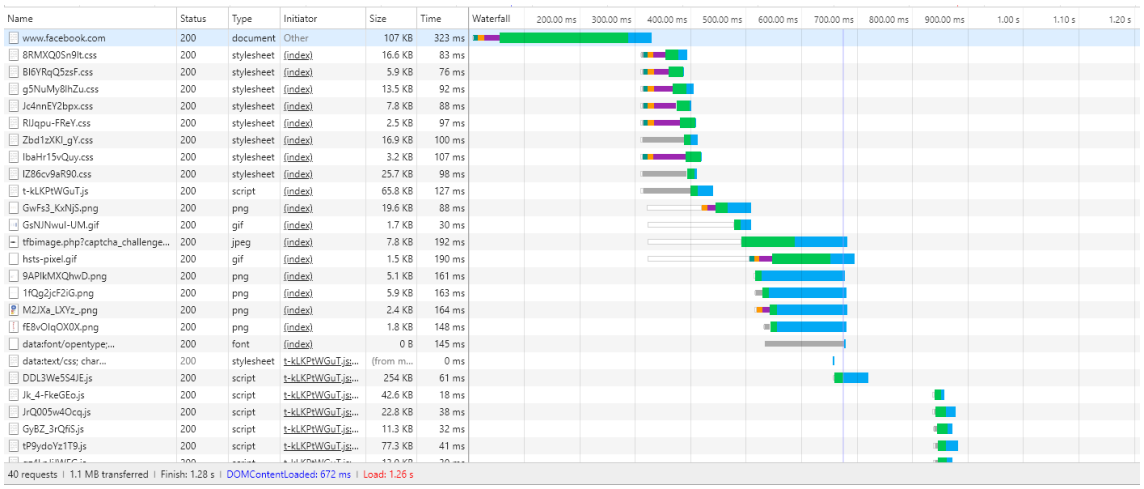


Figura 22 Facebook HTTP/1.1

El tiempo de carga medio fue de 672 ms mientras que el tiempo total para cargar la página al completo fue de 1260 ms.

Finalmente, para comprobar que esta página estaba usando http/1.1 se abrió un recurso para ver la estructura de la petición y la respuesta:

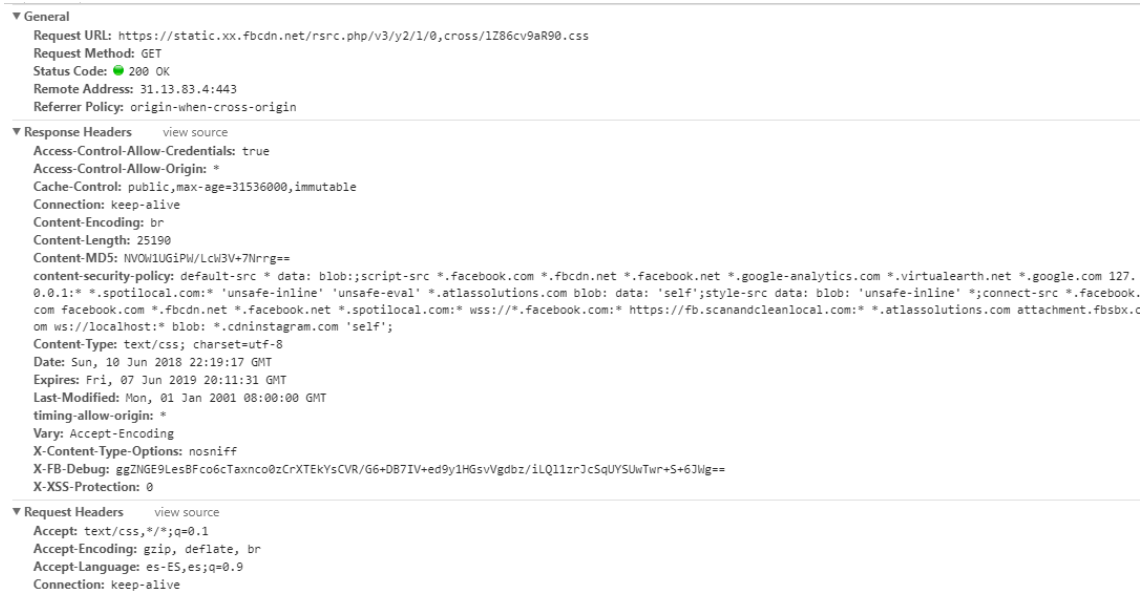
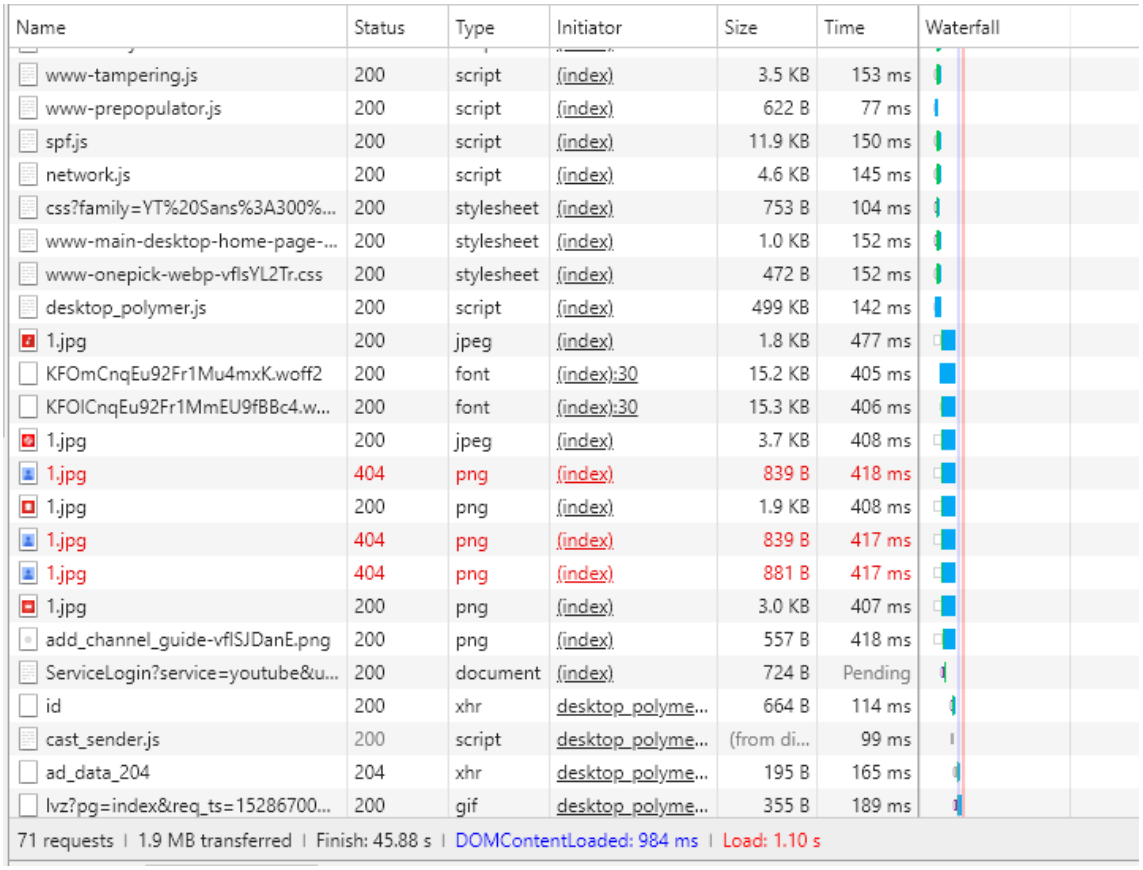


Figura 23 Comprobación Facebook HTTP/1.1

Una vez terminados las pruebas sobre HTTP/1.1, se procedió a vaciar de nuevo los datos de navegación, como se ha visto anteriormente, y se ejecutó Chrome sin ningún argumento y se volvieron a cargar las 3 páginas con la herramienta para desarrolladores.

YouTube tuvo los siguientes resultados en HTTP/2:



Name	Status	Type	Initiator	Size	Time	Waterfall
www-tampering.js	200	script	(index)	3.5 KB	153 ms	
www-prepopulator.js	200	script	(index)	622 B	77 ms	
spf.js	200	script	(index)	11.9 KB	150 ms	
network.js	200	script	(index)	4.6 KB	145 ms	
css?family=YT%20Sans%3A300%...	200	stylesheet	(index)	753 B	104 ms	
www-main-desktop-home-page-...	200	stylesheet	(index)	1.0 KB	152 ms	
www-onepick-webp-vflsYL2Tr.css	200	stylesheet	(index)	472 B	152 ms	
desktop_polymer.js	200	script	(index)	499 KB	142 ms	
1.jpg	200	jpeg	(index)	1.8 KB	477 ms	
KFOmCnqEu92Fr1Mu4mxK.woff2	200	font	(index):30	15.2 KB	405 ms	
KFOlCnqEu92Fr1MmEU9fBBc4.w...	200	font	(index):30	15.3 KB	406 ms	
1.jpg	200	jpeg	(index)	3.7 KB	408 ms	
1.jpg	404	png	(index)	839 B	418 ms	
1.jpg	200	png	(index)	1.9 KB	408 ms	
1.jpg	404	png	(index)	839 B	417 ms	
1.jpg	404	png	(index)	881 B	417 ms	
1.jpg	200	png	(index)	3.0 KB	407 ms	
add_channel_guide-vflSJDanE.png	200	png	(index)	557 B	418 ms	
ServiceLogin?service=youtube&u...	200	document	(index)	724 B	Pending	
id	200	xhr	desktop_polyme...	664 B	114 ms	
cast_sender.js	200	script	desktop_polyme...	(from di...	99 ms	
ad_data_204	204	xhr	desktop_polyme...	195 B	165 ms	
lvz?pg=index&req_ts=15286700...	200	gif	desktop_polyme...	355 B	189 ms	

71 requests | 1.9 MB transferred | Finish: 45.88 s | DOMContentLoaded: 984 ms | Load: 1.10 s

Figura 24 YouTube HTTP/2

El tiempo de carga medio fue de 984 ms mientras que el tiempo total para cargar la página al completo fue de 1100 ms.

Finalmente, para comprobar que esta página estaba usando HTTP/2 se abrió un recurso para ver la estructura de la petición y la respuesta, como se aprecia en la Figura 25, los campos en las peticiones y en las respuestas siguen el formato de HTTP/2 con los dos puntos en las cabeceras

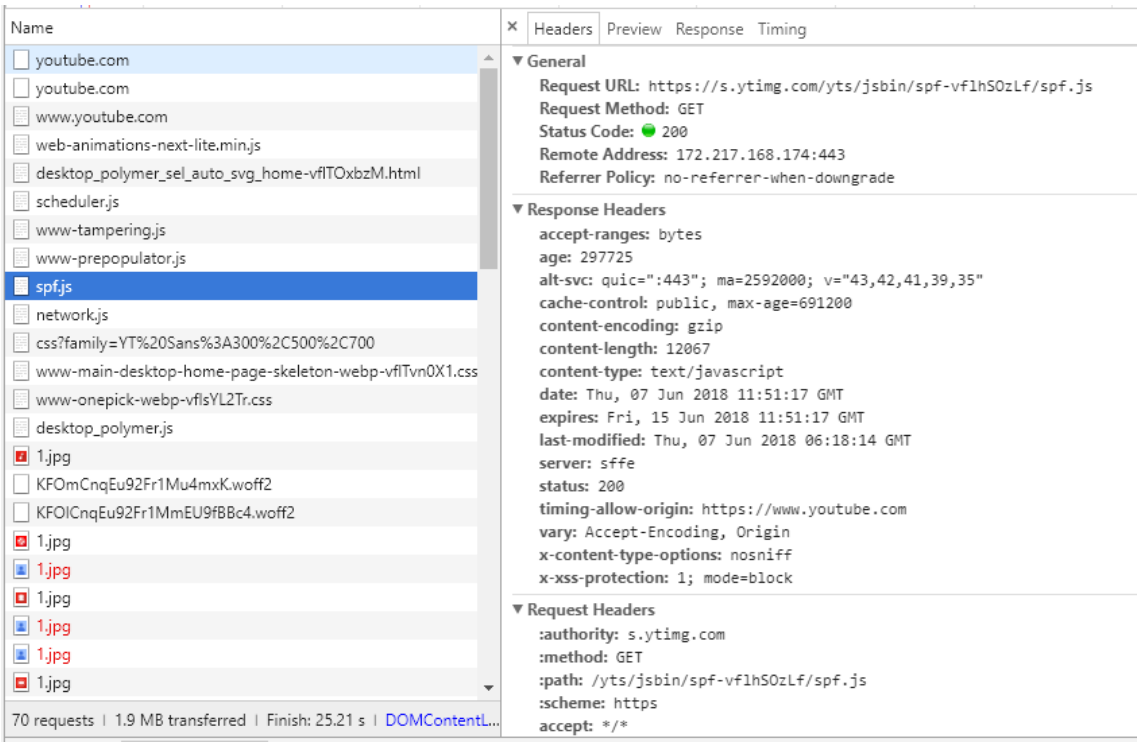


Figura 25 Comprobación YouTube HTTP/2

Twitter fue la página que se analizó en segundo lugar.

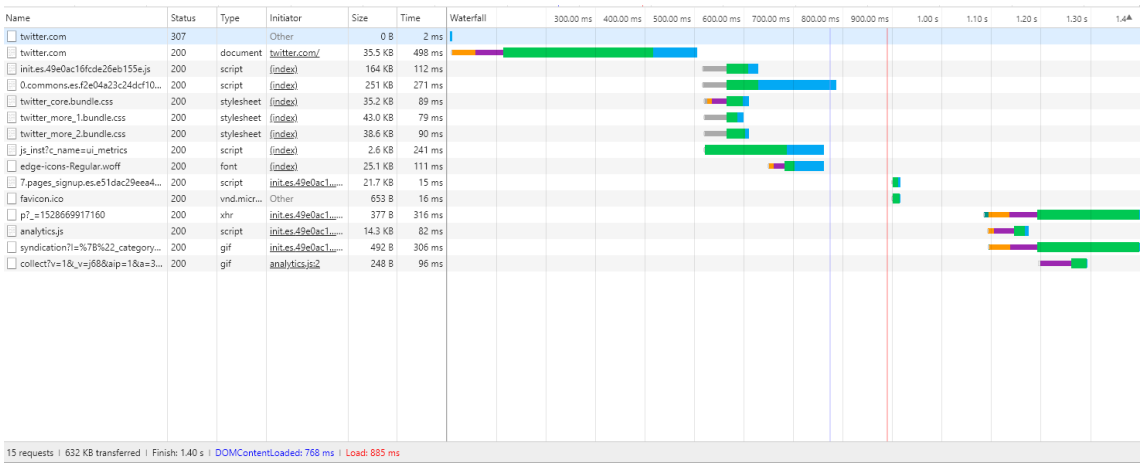


Figura 26 Twitter HTTP/2

El tiempo de carga medio fue de 768 ms mientras que el tiempo total para cargar la página al completo fue de 885 ms.

Finalmente, para comprobar que esta página estaba usando http/2 se abrió un recurso para ver la estructura de la petición y la respuesta:

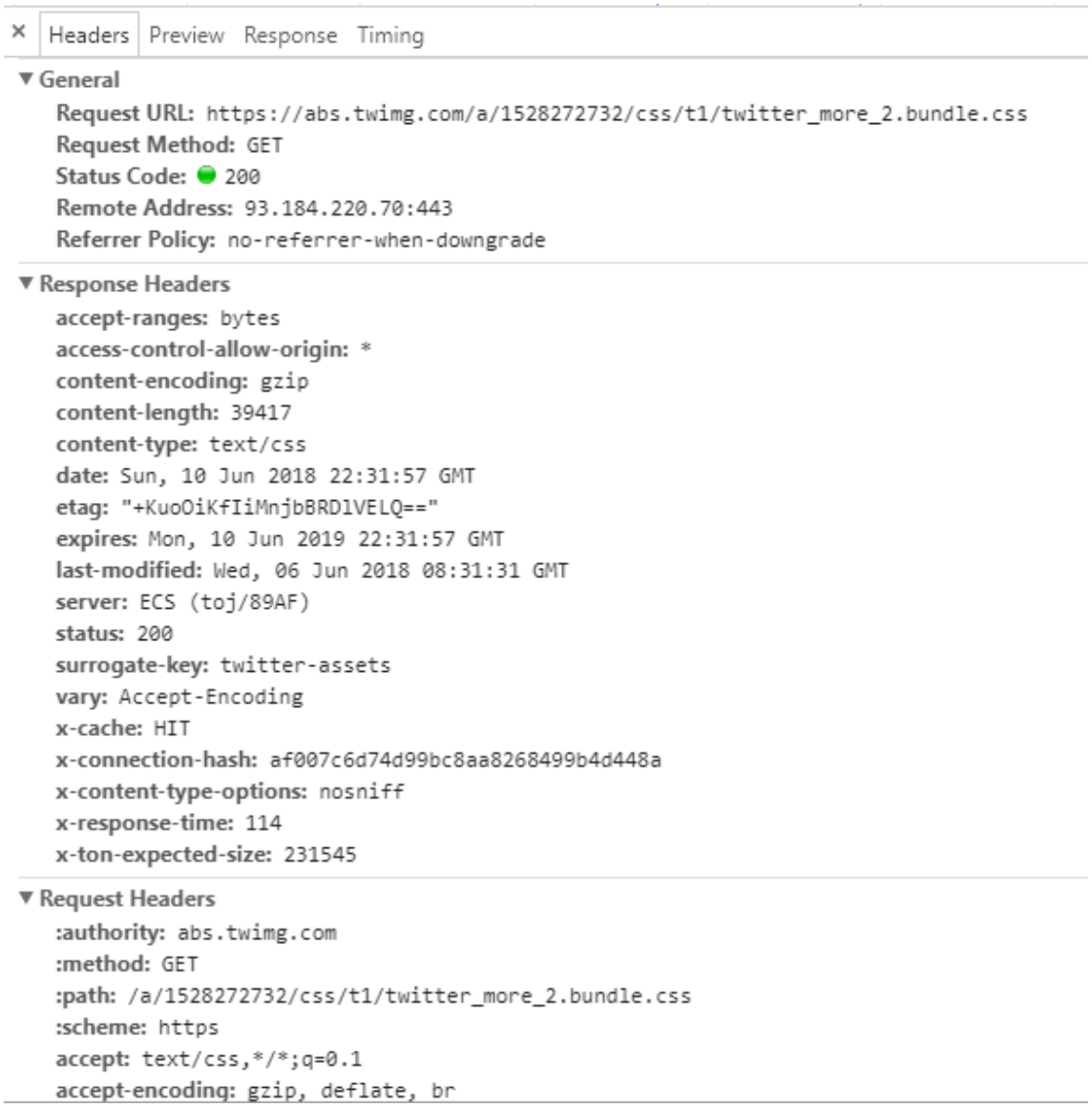


Figura 27Comprobación Twitter HTTP/2

Para finalizar con esta prueba se analizó Facebook, con los siguientes resultados:

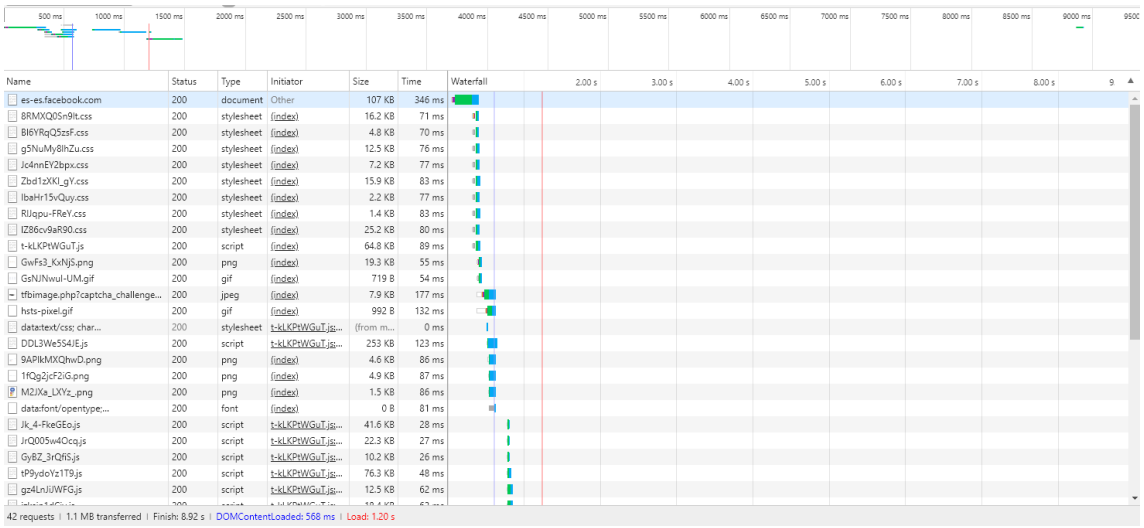
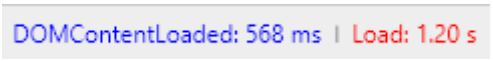


Figura 28 Facebook HTTP/2



El tiempo de carga medio fue de 568 ms mientras que el tiempo total para cargar la página al completo fue de 1200 ms.

Finalmente, para comprobar que esta página estaba usando http/2 se abrió un recurso para ver la estructura de la petición y la respuesta:

Figura 29 Comprobación Facebook HTTP/2

En la siguiente tabla se reúnen los datos obtenidos, cabe mencionar que los resultados obtenidos se corresponden a la mediana de 5 tests realizados a las páginas web, los resultados diferían aproximadamente en 1 décima de segundo como máximo.:

Tabla 5 Análisis Real Fibra Óptica

Página	Tiempo de carga medio (ms)		Tiempo para visualizar al completo la página (ms)	
	HTTP/2	HTTP/1.1	HTTP/2	HTTP/1.1
YouTube	984	882	1100	989
Twitter	768	704	885	815
Facebook	568	672	1200	1260

En este estudio realizado mediante una conexión de fibra óptica, podemos observar que los tiempos para la carga del documento base y de la carga completa de las páginas web de Twitter y YouTube son peores con HTTP/2, mientras que Facebook mejora estos tiempos. Cabe destacar que los tiempos obtenidos oscilan en una décima de segundo como se ha especificado a la hora de realizar este estudio y que es la mediana de los 5 test realizados y puesto que las diferencias obtenidas no superan esta décima de segundo, es probable que la diferencia no sea apreciable en ninguna web.

5.2.3 Demo Akamai

HTTP/2 vs HTTP/1.1

Aparte de los test realizados anteriormente, hay algunas demos por Internet que tratan de establecer una comparativa entre los protocolos HTTP/2 y HTTP/1.1.

Un ejemplo puede ser el que realiza Akamai, que se trata de una corporación que distribuye servicios globalmente en Internet, en su página web:

https://http2.akamai.com/demo

El funcionamiento es bastante sencillo, se trata de un HTML que contiene 2 imágenes, divididas a su vez en varias imágenes, cada una formando un bloque de pixeles, que se descarga en HTTP/1.1:

tile-0.png	200	png	h2_demo_frame....	1.4 KB	113 ms
tile-1.png	200	png	h2_demo_frame....	1.4 KB	112 ms
tile-2.png	200	png	h2_demo_frame....	1.5 KB	59 ms
tile-3.png	200	png	h2_demo_frame....	1.4 KB	130 ms
tile-4.png	200	png	h2_demo_frame....	1.3 KB	130 ms
tile-5.png	200	png	h2_demo_frame....	1.4 KB	154 ms
f7c4f810-ae5e-4a89-94d1-c0f508...	200	script	inject.preload.js:...	(from di...	12 ms
aksb.min.js	200	script	h2_demo_frame....	(from di...	32 ms
tile-6.png	200	png	h2_demo_frame....	1.4 KB	1.87 s
tile-50.png	200	png	h2_demo_frame....	1.3 KB	103 ms
tile-104.png	200	png	h2_demo_frame....	4.3 KB	102 ms
tile-20.png	200	png	h2_demo_frame....	1.3 KB	89 ms
tile-38.png	200	png	h2_demo_frame....	1.4 KB	100 ms
tile-47.png	200	png	h2_demo_frame....	1.3 KB	89 ms
tile-26.png	200	png	h2_demo_frame....	2.1 KB	99 ms
tile-51.png	200	png	h2_demo_frame....	1.3 KB	99 ms
tile-88.png	200	png	h2_demo_frame....	3.3 KB	100 ms
tile-35.png	200	png	h2_demo_frame....	1.4 KB	100 ms

Figura 30 Akamai HTTP/1.1

Cada fragmento de la imagen se descarga del servidor de HTTP/1.1 y se especifica que no se puede almacenar en cache, para poder realizar el test constantemente.

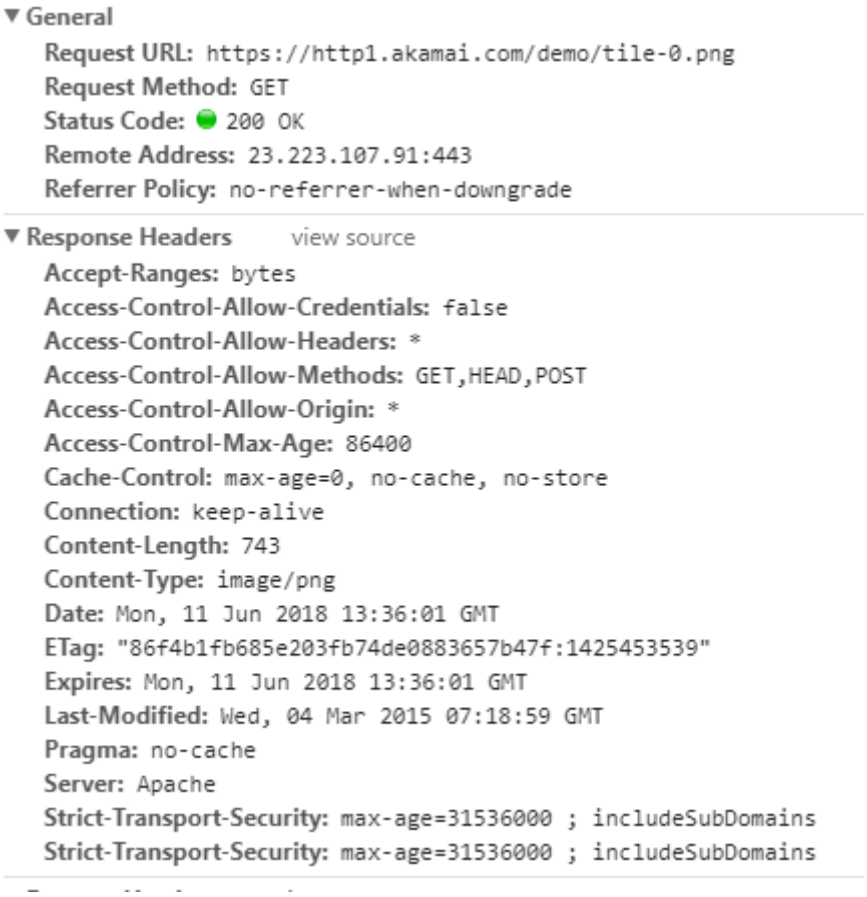


Figura 31 Comprobación Akamai HTTP/1.1

Además, al finalizar se especifica la latencia que existe al servidor de HTTP/1.1, puesto que es distinto al de HTTP/2 y se registra el tiempo que ha costado cargar la imagen.

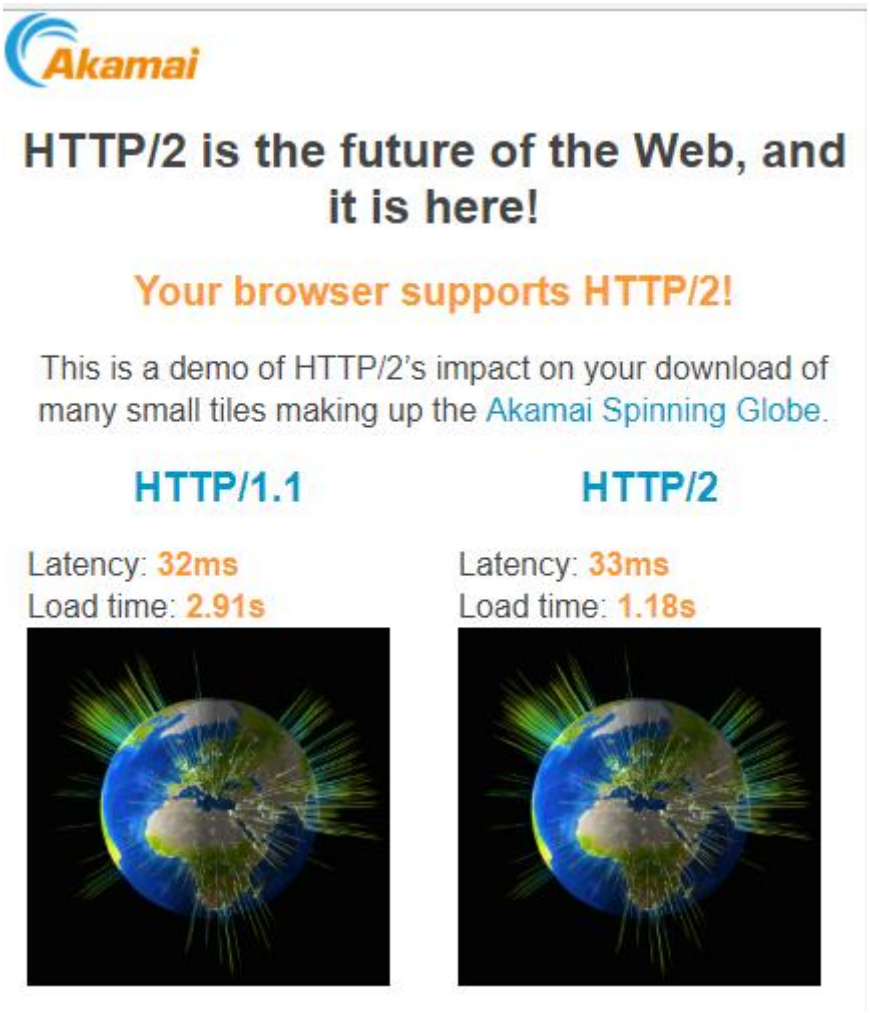


Figura 32 Akamai HTTP/1.1 vs HTTP/2

A continuación, se procede a cargar la página en HTTP/2, para ello, se empieza por detectar si el navegador web utiliza HTTP/2, tal y como se especifica en la descripción de esta demostración. Si el navegador no es compatible, se muestra un mensaje explicando que HTTP/2 no está activo y que para ver esta demostración es necesario que se visite la página web con un navegador que sí que lo pueda visualizar.

Una vez realizado este primer paso, el procedimiento es igual a HTTP/1.1, se descargan las 379 imágenes mediante el servidor de HTTP/2 y se especifica que no se puede almacenar en cache, para poder repetir el test constantemente.

tile-378.png	200	png	h2_demo_frame....	1.4 KB	20 ms
h2_demo_frame.html	200	document	demo:167	38.8 KB	276 ms
tile-0.png	200	png	h2_demo_frame....	1.1 KB	197 ms
tile-1.png	200	png	h2_demo_frame....	1.1 KB	197 ms
tile-2.png	200	png	h2_demo_frame....	1.1 KB	198 ms
tile-3.png	200	png	h2_demo_frame....	1.1 KB	198 ms
tile-4.png	200	png	h2_demo_frame....	1.1 KB	198 ms
tile-5.png	200	png	h2_demo_frame....	1.1 KB	212 ms
tile-6.png	200	png	h2_demo_frame....	1.1 KB	236 ms
tile-7.png	200	png	h2_demo_frame....	1.1 KB	241 ms
tile-8.png	200	png	h2_demo_frame....	1.2 KB	262 ms
tile-9.png	200	png	h2_demo_frame....	1.1 KB	289 ms
tile-10.png	200	png	h2_demo_frame....	1.1 KB	293 ms
tile-11.png	200	png	h2_demo_frame....	1.1 KB	310 ms
tile-12.png	200	png	h2_demo_frame....	1.1 KB	310 ms
tile-13.png	200	png	h2_demo_frame....	1.1 KB	313 ms
tile-14.png	200	png	h2_demo_frame....	1.1 KB	315 ms
tile-15.png	200	png	h2_demo_frame....	1.1 KB	316 ms
tile-16.png	200	png	h2_demo_frame....	1.1 KB	317 ms

Figura 33 Akamai HTTP/2

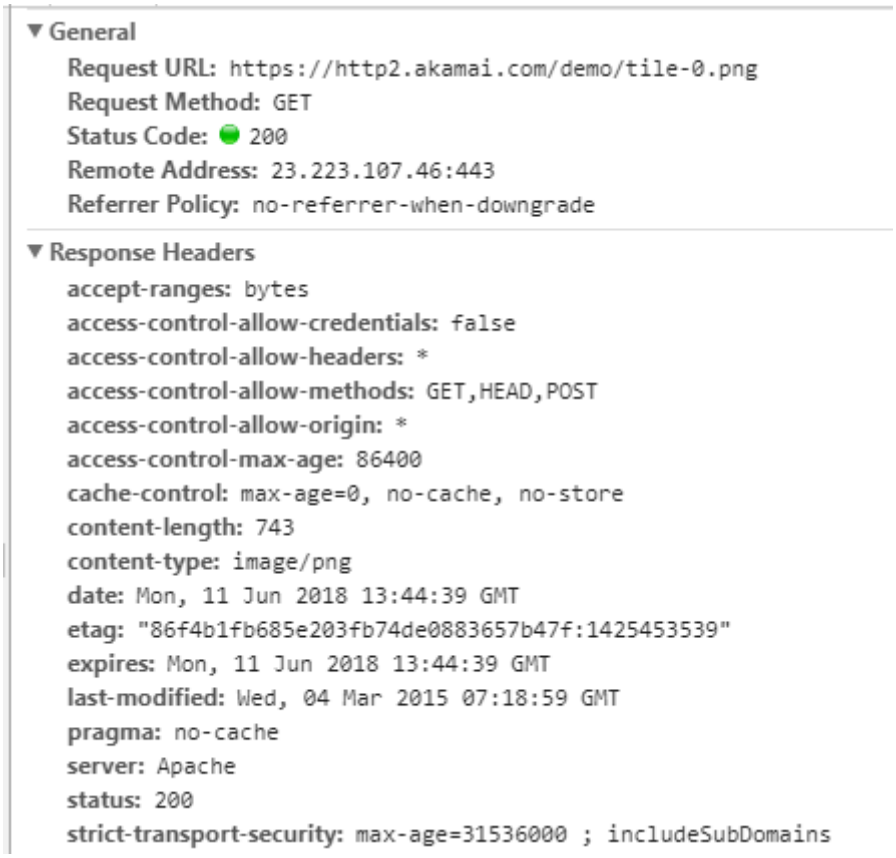


Figura 34 Comprobación Akamai HTTP/2

Una última comprobación, para ver que se ha conectado efectivamente con HTTP/2 es visitar la URL de Chrome:

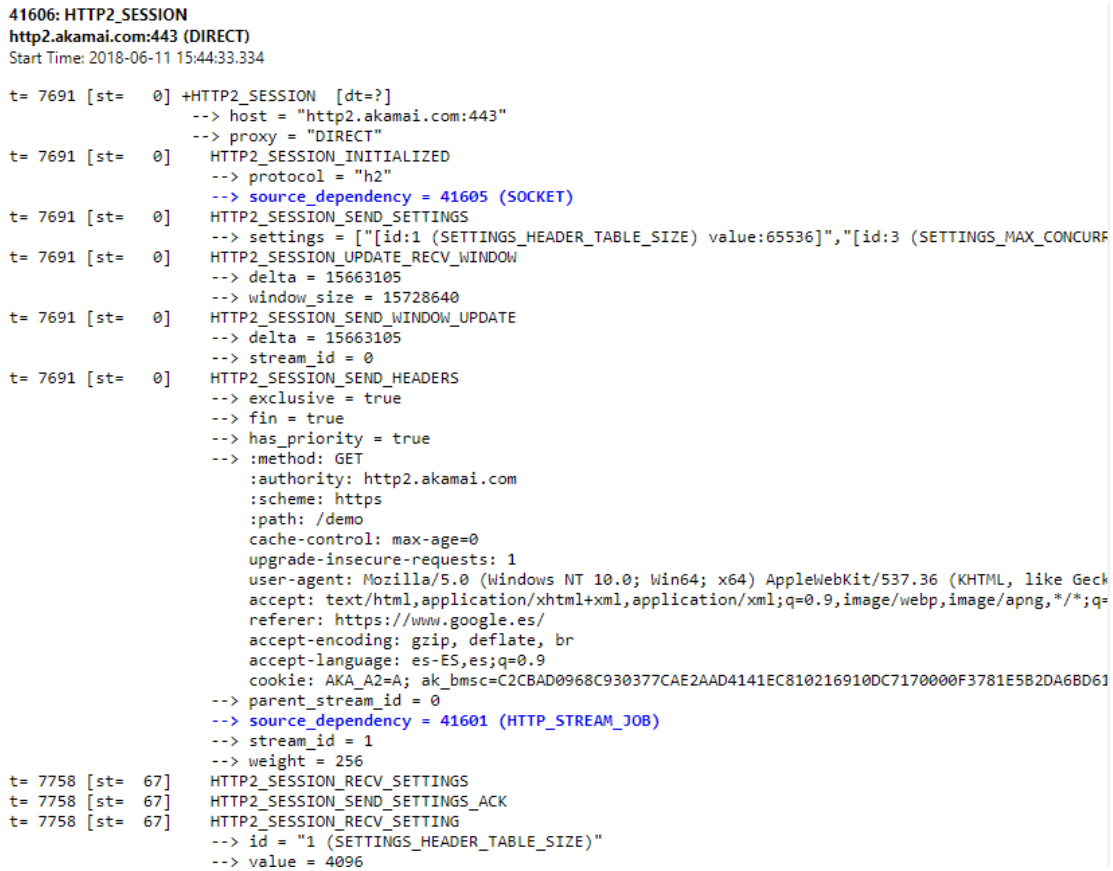
chrome://net-internals/#events&q=type:HTTP2_SESSION%20is:active

Que nos muestra qué sesiones de HTTP/2 están activas.

Finalmente se observa que está ahí la sesión de Akamai:

Figura 35 Sesión HTTP2 1/2

Con sus correspondientes tramas:



```
41606: HTTP2_SESSION
http2.akamai.com:443 (DIRECT)
Start Time: 2018-06-11 15:44:33.334

t= 7691 [st=   0] +HTTP2_SESSION  [dt=?]
                   --> host = "http2.akamai.com:443"
                   --> proxy = "DIRECT"
t= 7691 [st=   0]    HTTP2_SESSION_INITIALIZED
                     --> protocol = "h2"
                     --> source_dependency = 41605 (SOCKET)
t= 7691 [st=   0]    HTTP2_SESSION_SEND_SETTINGS
                     --> settings = ["[id:1 (SETTINGS_HEADER_TABLE_SIZE) value:65536]","[id:3 (SETTINGS_MAX_CONCURR
t= 7691 [st=   0]    HTTP2_SESSION_UPDATE_RECV_WINDOW
                     --> delta = 15663105
                     --> window_size = 15728640
t= 7691 [st=   0]    HTTP2_SESSION_SEND_WINDOW_UPDATE
                     --> delta = 15663105
                     --> stream_id = 0
t= 7691 [st=   0]    HTTP2_SESSION_SEND_HEADERS
                     --> exclusive = true
                     --> fin = true
                     --> has_priority = true
                     --> :method: GET
                         :authority: http2.akamai.com
                         :scheme: https
                         :path: /demo
                         cache-control: max-age=0
                         upgrade-insecure-requests: 1
                         user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
                         accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
                         referer: https://www.google.es/
                         accept-encoding: gzip, deflate, br
                         accept-language: es-ES,es;q=0.9
                         cookie: AKA_A2=A; ak_bmsc=C2CBAD0968C930377CAE2AAD4141EC810216910DC7170000F3781E5B2DA6BD61
                     --> parent_stream_id = 0
                     --> source_dependency = 41601 (HTTP_STREAM_JOB)
                     --> stream_id = 1
                     --> weight = 256
t= 7758 [st=  67]    HTTP2_SESSION_RECV_SETTINGS
t= 7758 [st=  67]    HTTP2_SESSION_SEND_SETTINGS_ACK
t= 7758 [st=  67]    HTTP2_SESSION_RECV_SETTING
                     --> id = "1 (SETTINGS_HEADER_TABLE_SIZE)"
                     --> value = 4096
```

Figura 36 Sesión HTTP/2 2/2

Los tiempos obtenidos como se observan en la figura 32, son de 2.91 segundos para HTTP/1.1 mientras que HTTP/2 cargo la foto en 1.18 segundos.

HTTP/2 vs HTTP/2 con Server Push

Una vez analizado el comportamiento de HTTP/2 en distintas páginas web, apenas se realizaban push de objetos en las páginas que usaban este protocolo, como se muestra en la siguiente figura:

HTTP/2 sessions

View live HTTP/2 sessions

Host	Proxy	ID	Negotiated Protocol	Active streams	Unclaimed pushed	Max	Initiated	Pushed
assets.gyazo.com:443	direct://	532268	h2	0	0	256	6	0
cdn.mxpnl.com:443	direct://	532295	h2	0	0	100	1	0
cdnjs.cloudflare.com:443	direct://	532265	h2	0	0	256	2	0
goo.gl:443	direct://	532041	h2	0	0	100	1	0
gyazo.com:443	direct://	532211	h2	0	0	100	41	0
use.typekit.net:443 p.typekit.net:443	direct://	532266	h2	0	0	100	5	0
assets.gyazo.com:443 geo.gyazo.com:443 i.gyazo.com:443	direct://	532406	h2	0	0	256	9	0
cdn.ravenjs.com:443	direct://	532480	h2	0	0	100	1	0

Figura 37 Demostración server push

Como apenas existen páginas que hagan uso de esta herramienta, se procedió a estudiar otra demostración que ofrece Akamai.

Para acceder a esta demostración basta con entrar a la siguiente URL:

https://http2.akamai.com/demo/index_sp.html?num=30

En ella como se puede observar en la siguiente figura, se observa el tiempo que se tarda en obtener la misma foto que se ha visualizado en la anterior demostración, con los mismos recursos, la única diferencia es que existen 30 fragmentos, que se han decidido enviarlos previamente mediante el uso de Server Push, la diferencia de tiempos se muestra a continuación:

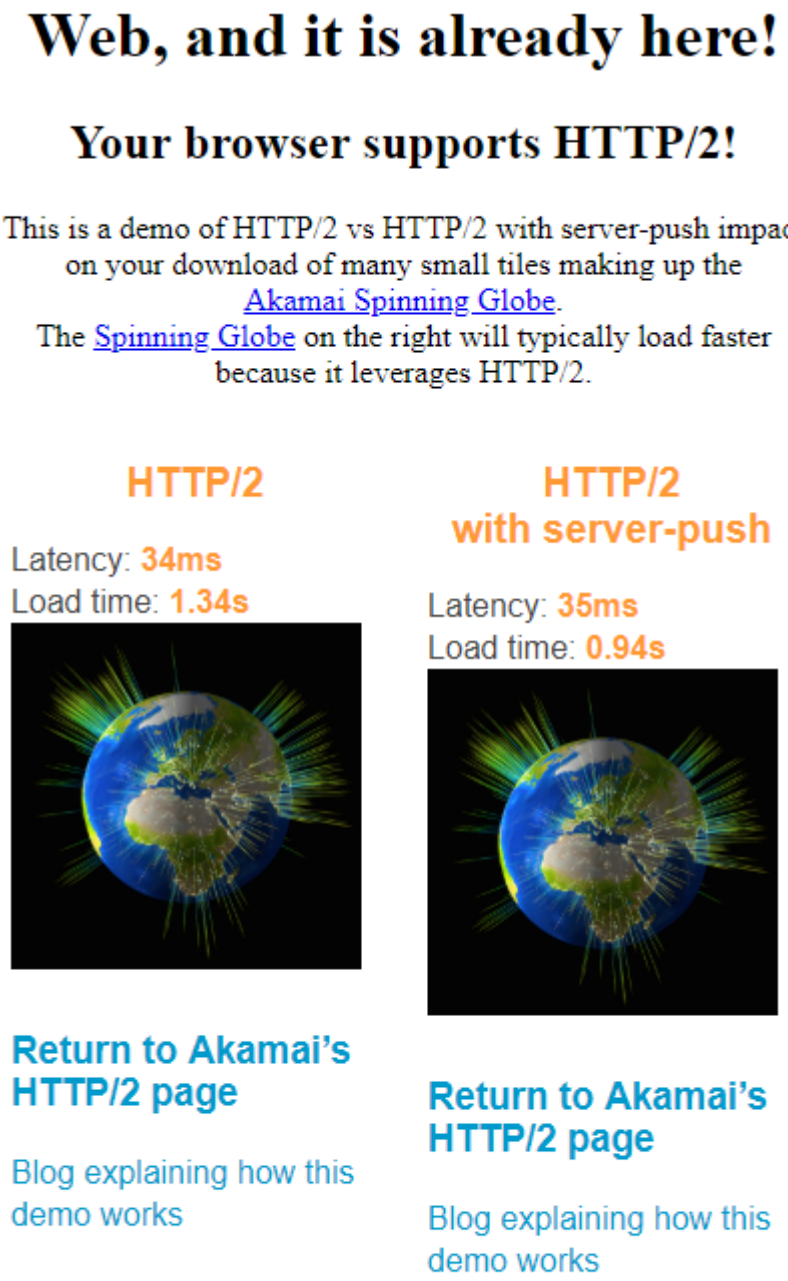


Figura 38 HTTP/2 vs HTTP/2 con Server Push

Como se puede observar el tiempo disminuye considerablemente, si se hace uso de esta característica de HTTP/2, algo que podría beneficiar mucho a las páginas web actuales debido a que son cuatro décimas de segundo la diferencia entre ambos tiempos, usando la característica de server push en 30 fragmentos.

Para comprobar que efectivamente se ha hecho uso de esta herramienta, se procede a la visualización mediante la herramienta para desarrolladores que ofrece Chrome.

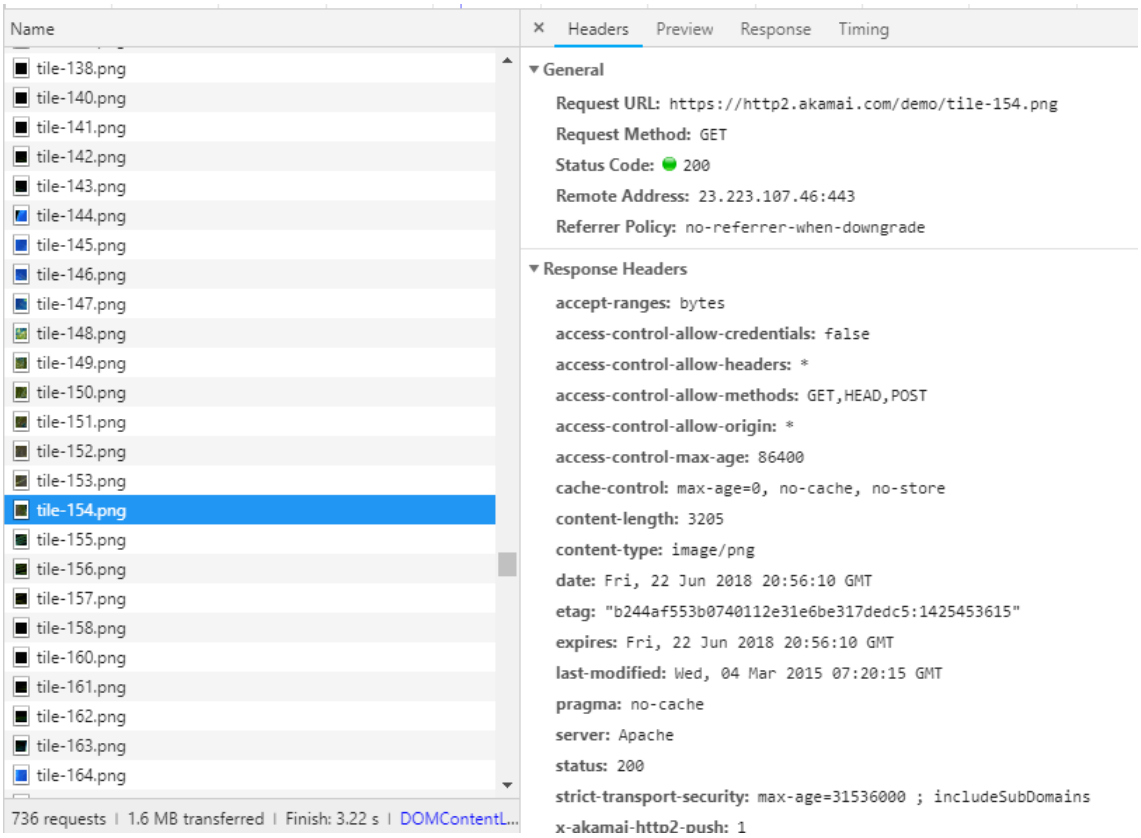


Figura 39 Demostración Push 1/3

Como podemos observar mediante las cabeceras, este fragmento no se puede guardar en cache, además es HTTP/2 por el uso de cabeceras en minúscula y como se puede apreciar a continuación, se ha obtenido mediante un Push:

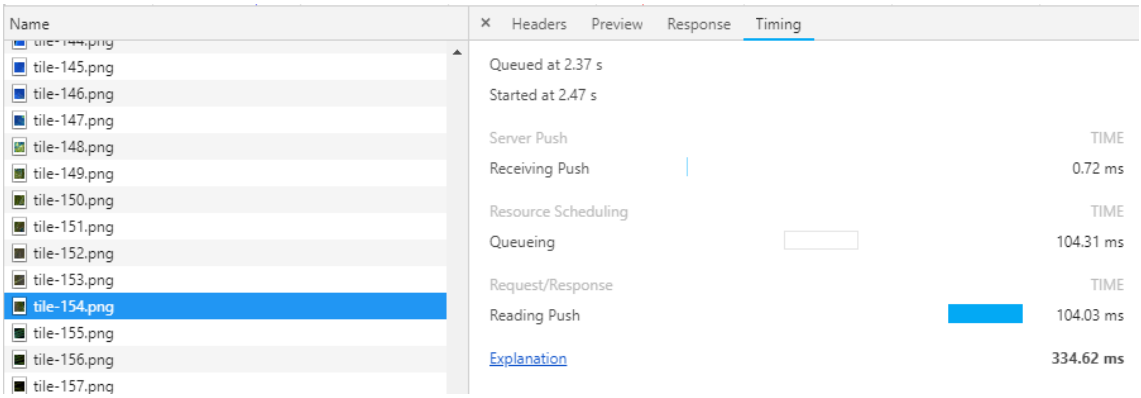


Figura 40 Demostración Push 2/3

Host	Proxy	ID	Negotiated Protocol	Active streams	Unclaimed pushed	Max	Initiated	Pushed	Pushed and claimed	A
cdn.mxpnl.com:443	direct://	534420	h2	0	0	100	1	0	0	0
gyazo.com:443	direct://	534380	h2	0	0	100	11	0	0	0
http2.akamai.com:443	direct://	534652	h2	1	1	100	695	33	30	0
clients4.google.com:443	direct://	534599	h2	0	0	100	2	0	0	0
i.gyazo.com:443 geo.gyazo.com:443	direct://	534547	h2	0	0	256	2	0	0	0

Figura 41 Demostración Push 3/3

Y para finalizar vemos que efectivamente Akamai usa esta característica.

La demostración realizada por Akamai, que muestra 2 imágenes , una descargada con HTTP/1.1 y otra con HTTP/2, el tiempo es mucho mejor en HTTP/2 que en HTTP/1.1, pero hay que tener en cuenta varias cosas, las imágenes descargadas son 379, algo que cualquier página web, normalmente no tiene, puesto que HTTP/1.1 solo puede tener 6 conexiones simultaneas, descargar 379 imágenes genera un tiempo de espera que no se realiza en HTTP/2. Además, puesto que en HTTP/1.1 se tiene que esperar a tener una respuesta a cada petición, mientras que en HTTP/2 no es necesario, y, además, es importante observar como los tamaños de las descargas, pese a que los bytes son iguales en el content-length de la petición del pixel 0, difieren en 0.3 KB, como se observa en la cascada de conexión.

Por otro lado, el empleo de server push, algo apenas usado por las páginas web, mejora considerablemente el tiempo para cargar la página web, es difícil saber que objetos pueden ser enviados mediante este método, pero la mejora de cuatro décimas de segundo supone una gran mejora que puede ser utilizada.

6. Conclusiones

HTTP/2 es un protocolo que acaba de salir a la luz, es por lo que su comportamiento puede no ser el deseado. Si bien es cierto que hace un mejor uso del protocolo TCP, es necesario recalcar que una página web necesita revertir los cambios realizados en HTTP/1.1 que pueden ser incoherentes con la estructura de HTTP/2, tales como la concatenación, la fragmentación o el "spriting".

Teniendo en cuenta estas consideraciones, al analizar 3 de las páginas web más visitadas del mundo como son Facebook, Twitter y YouTube, se puede observar que la estructura de cada página web condiciona el funcionamiento de HTTP/2.

Es importante establecer que como se ha visto anteriormente, hay páginas que no se benefician de HTTP/2 y este estudio se ha realizado pensando en el navegador que más diferencias mostraba, puesto que la implementación cuenta.

Por tanto, podemos concluir bajo estos test realizados, que Facebook ha hecho grandes mejoras en sus páginas web, para la adaptación a este nuevo protocolo, puesto que la gran mayoría de pruebas obtienen mejores tiempos bajo HTTP/2, mientras que Twitter y YouTube no presentan una mejora, puesto que Twitter siempre obtiene resultados similares y YouTube obtiene resultados mejores y peores.

Para concluir este estudio sobre el protocolo HTTP/2, es necesario entender que pese a que las mejoras actualmente no son muchas, e incluso algunas páginas empeoran su rendimiento, HTTP/2 es mejor protocolo que HTTP/1.1, puesto que cuando existen perdidas de paquetes, se produce el efecto conocido como HOL blocking, que se soluciona mejor en HTTP/2, además actualmente las páginas pueden tener algunas imágenes, texto y algún fichero de JavaScript, pero cuando se necesitan muchas imágenes en una página, como por ejemplo la página web de Imágenes de Google, al estar implementada sobre HTTP/2 y no usar push, se puede hacer uso del Server Push para enviar las imágenes siguientes, y así evitar tiempos de espera a que se descarguen las imágenes, además se puede realizar en paralelo, sin tener que esperar a respuestas anteriores que pueden bloquear una conexión. En el futuro como se pretende usar más recursos en las páginas web y que sean más sofisticadas, este protocolo mejora el rendimiento al descargar en paralelo y usar la multiplexación.

Como pasó con HTTP/1.1, en sus inicios no había grandes ventajas en su utilización, pero con las mejoras que se fueron haciendo progresivamente, hicieron que su eficiencia mejorase considerablemente.

Cabe resaltar que Google pese a ser el principal propulsor de HTTP/2 creando SPDY, no proporciona una mejora sustancial en sus páginas web como se puede observar en YouTube, cosa que hace Facebook.

Aparte de este protocolo, cabe resaltar que existe un nuevo protocolo llamado QUIC, desarrollado por Google y basado en UDP que puede mejorar los tiempos de HTTP/2. Pero ya se observará en un futuro si el cambio hacia ese protocolo resulta beneficioso o no.

7. Bibliografía

[1] LUDIN, S.; GARZA, J. (2016). Learning HTTP/2: An Introduction to the Next Generation Web. O'Reilly

[2] KUROSE, James F.; ROSS, Keith W. (2013). Computer networking: a top-down approach. *Addison Wesley Computing*,

[3] GRIGORIK, Ilya. (2013). *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.",

[4] BELSHE, M; THOMSON, M; PEON, R. (2015). RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). <https://tools.ietf.org/html/rfc7540/ > [Consulta: 8 de mayo 2018]

[5] Webpagetest (website performance and optimization test), <https://www.webpagetest.org >/. [Consulta: 15 de mayo 2018]

[6] COMBS, Gerald, Wireshark – Go Deep<https://www.wireshark.org/> [Consulta: 20 de mayo 2018]

[7] The OpenSSL Project, OpenSSL <https://www.openssl.org/> [Consulta: 10 de mayo 2018]

[8] TSUJIKAWA, T. Nghttp2: Http/2 c library. 2015. <https://nghttp2.org/> [Consulta: 20 de mayo 2018]

[9] Juskenas, A. (2016)." Bye raster, hello vector: 3 ways to use SVG easier" *Devbrige Group* <https://www.devbridge.com/articles/bye-raster-hello-vector-3-ways-to-use-svg-easier-1/> [Consulta: 4 de junio de 2018]

[10] Comprar y Vender Electrónica, Moda, Móviles y mucho más | eBay <https://www.ebay.es/> [Consulta: 14 de mayo de 2018]

[11] Can I use... Support tables for HTML5, CSS3, etc <https://caniuse.com/#search=http2 > [Consulta: 17 de mayo de 2018]

[12] Pollard, B. (2018). *HTTP/2 in Action*. Manning Publications

[13] Android Developers. < https://developer.android.com/studio/debug/dev-options> [Consulta 10 de junio de 2018]

[14] Shaver, J. (2015)." Decrypting TLS Browser Traffic With Wireshark – The Easy Way!" < https://jimshaver.net/2015/02/11/decrypting-tls-browser-traffic-with-wireshark-the-easy-way/> [Consulta 8 de junio de 2018]

[15] NOTTINGHAM, M; MCMANUS, P; RESCHKE, J. (2016). RFC 7838 - HTTP ALTERNATIVE SERVICES. <https://tools.ietf.org/html/rfc7838 > [Consulta: 27 de mayo 2018]

[16] FRIEDL, S; POPOV, A; LANGLEY, A; STEPHAN, E. (2014). RFC 7301 - Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. <https://tools.ietf.org/html/rfc7301 > [Consulta: 30 de mayo 2018]

[17] FIELDING, R; GETTYS, J; MOGUL, J; FRYSTYK, H; MASINTER, L; LEACH, P; BERNERS-LEE, T. (1999). RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1 <https://tools.ietf.org/html/rfc2616> [Consulta: 1 de junio 2018]

8. Anexos

Anexo A Tramas de HTTP/2

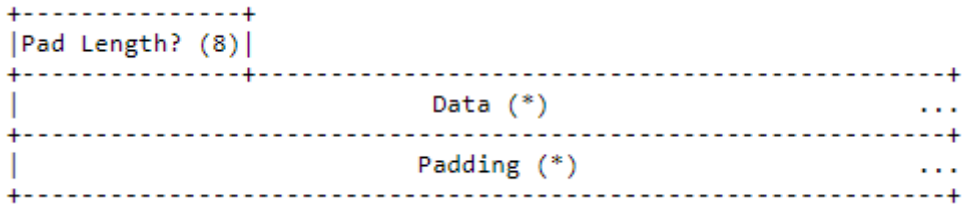
Como se ha explicado en el capítulo 3 de este documento, HTTP/2 tiene 10 tipos de tramas, identificadas cada una mediante un código de 8 bits único.

Estos bits se establecen en el campo Type de la cabecera de la trama, para ver la cabecera de la trama ver Página 27.

DATA

La trama Data contiene una cantidad variable de secuencias de octetos asociados a un flujo. Normalmente se utilizan para transmitir las peticiones y respuestas en HTTP/2. Su valor en el campo Type es 0x0.

Adicionalmente se puede encontrar el "padding" para camuflar el tamaño de los mensajes.



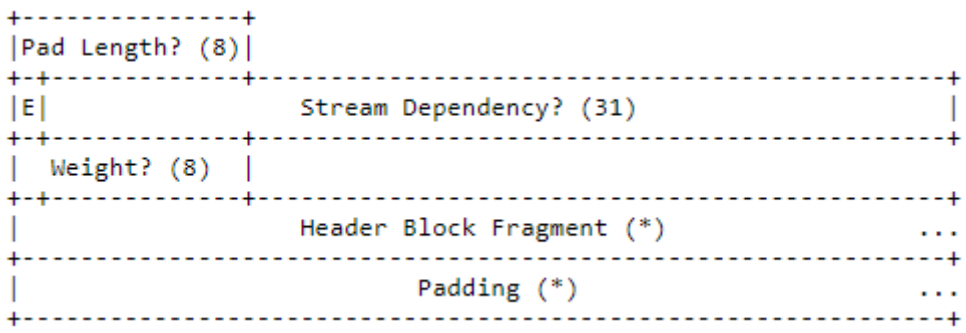
Campo	Descripción
Pad length	Indica el tamaño del Padding mediante 8 bits, es opcional y solo se transmite si el flag PADDED esta activado.
Data	Son los datos transmitidos en la trama
Padding	Son ceros enviados que no contienen ningún valor semántico, sirven para camuflar el tamaño de los mensajes y su tamaño lo indica la cabecera anterior, Pad Length.

Flag	Descripción
END_STREAM	Es el bit 0x1, cuando se activa, indica que es la ultima trama que enviara con ese identificador de flujo
PADDED	Es el bit 0x8, cuando se activa, indica que los campos Pad Length y Padding están presentes

Las tramas DATA deben estar asociadas con un flujo, si recibe una trama con el identificador 0x0, se debe responder con error de conexión PROTOCOL_ERROR.

HEADERS

La trama Headers se utiliza para iniciar un flujo. Su valor en el campo Type es 0x1.



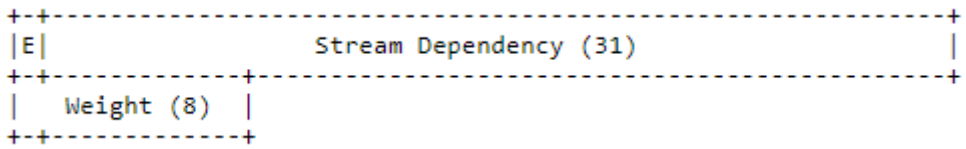
Campo	Descripción
Pad length	Indica el tamaño del Padding mediante 8 bits, es opcional y solo se transmite si el flag PADDED esta activado.
E	Un flag que indica que la dependencia de flujo es exclusiva. Solo está presente si el flag de PRIORITY está activado.
Stream Dependency	Un valor de 31 bits que identifica el flujo del cual depende. Solo está presente si el flag de PRIORITY está activado.
Weight	Un valor entero de 8 bits sin signo que representa el peso asociado al flujo. Solo está presente si el flag de PRIORITY está activado.
Header Block Fragment	Son las cabeceras del mensaje.
Padding	Son ceros enviados que no contienen ningún valor semántico, sirven para camuflar el tamaño de los mensajes y su tamaño lo indica la cabecera anterior, Pad Length.

Flag	Descripción
END_STREAM	Es el bit 0x1, cuando se activa, indica que es la última trama que enviara con ese identificador de flujo
END_HEADERS	Es el bit 0x4, cuando se activa, indica que contiene un bloque de cabeceras completo y no contiene ninguna trama Continuation
PADDED	Es el bit 0x8, cuando se activa, indica que los campos Pad Length y Padding están presentes
PRIORITY	Es el bit 0x20, cuando se activa, indica que los campos E, Stream Dependency y Weight están en la trama.

Las tramas Headers deben estar asociadas con un flujo, si recibe una trama con el identificador 0x0, se debe responder con error de conexión PROTOCOL_ERROR.

PRIORITY

La trama Priority se utiliza para indicar la prioridad del flujo. Se puede enviar muchas veces para cambiar las prioridades anteriores. Su valor en el campo Type es 0x2.

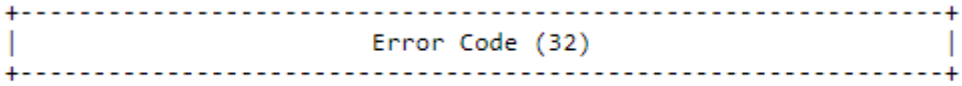


Campo	Descripción
E	Un flag que indica que la dependencia de flujo es exclusiva.
Stream Dependency	Un valor de 31 bits que identifica el flujo del cual depende.
Weight	Un valor entero de 8 bits sin signo que representa el peso asociado al flujo.

Las tramas Priority deben estar asociadas con un flujo, si recibe una trama con el identificador 0x0, se debe responder con error de conexión PROTOCOL_ERROR. Además, no define ningún flag.

RST_STREAM

La trama Rst_Stream se utiliza para solicitar la cancelación de un flujo o que una condición de error ha ocurrido. Su valor en el campo Type es 0x3.

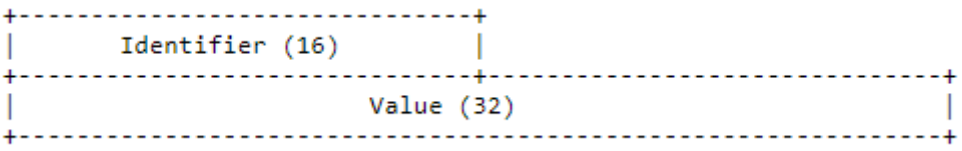


Campo	Descripción
Error Code	Un valor de 32 bits sin signo que identifica el código de error

Las tramas RST_STREAM deben estar asociadas con un flujo, si recibe una trama con el identificador 0x0, se debe responder con error de conexión PROTOCOL_ERROR. Además, no define ningún flag.

SETTINGS

La trama Settings se utiliza para configurar una comunicación entre 2 extremos. Cada trama consta de una pareja identificador y valor. Su valor en el campo Type es 0x4.



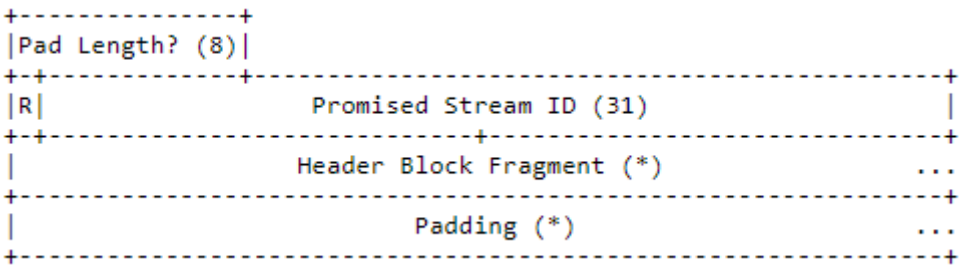
Campo	Descripción
Identifier	Indica el parámetro mediante un entero de 16 bits.
Value	Es el valor del parámetro, se expresa mediante 32 bits.

Parámetro	Identificador	Descripción
SETTINGS_HEADER_TABLE_SIZE	0x1	Permite al que envía la trama informar del tamaño máximo de la tabla usada en la compresión de cabeceras.
SETTINGS_ENABLE_PUSH	0x2	Permite deshabilitar la función "push" en el servidor. Su valor inicial es 1, que indica que está permitido.
SETTINGS_INITIAL_WINDOW_SIZE	0x3	Indica el número máximo de flujos concurrentes que el emisor permite enviar.
SETTINGS_INITIAL_WINDOW_SIZE	0x4	Indica el tamaño de ventana de un flujo. Su valor inicial es de 2^{16} -1 octetos. Valor máximo de 2^{31} -1.
SETTINGS_MAX_FRAME_SIZE	0x5	Indica el tamaño máximo que puede recibir el emisor en el payload.
SETTINGS_MAX_HEADER_LIST_SIZE	0x6	Informa del tamaño máximo de la lista de cabecera que el emisor está dispuesto a aceptar.

Las tramas Settings se envían al principio del prólogo de conexión, entre ambos extremos. Además, deben de retornar un flag de ACK (0x1) en la cabecera de la trama.

PUSH_PROMISE

La trama Push_Promise se utiliza para notificar al extremo, antes de transmitir el flujo, que tiene la intención de enviarlo. Su valor en el campo Type es 0x5.

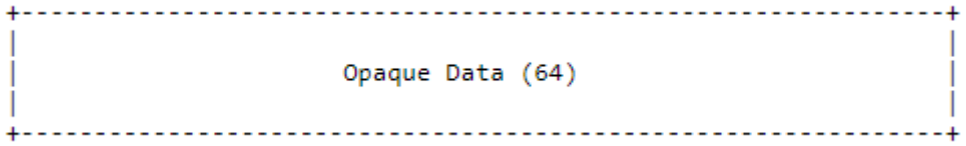


Campo	Descripción
Pad length	Indica el tamaño del Padding mediante 8 bits, es opcional y solo se transmite si el flag PADDED esta activado.
R	Un bit reservado.
Promised Stream ID	Un valor de 31 bits que identifica el flujo que está reservado por la trama.
Header Block Fragment	Son las cabeceras del mensaje.
Padding	Son ceros enviados que no contienen ningún valor semántico, sirven para camuflar el tamaño de los mensajes y su tamaño lo indica la cabecera anterior, Pad Length.

Flag	Descripción
END_HEADERS	Es el bit 0x4, cuando se activa, indica que contiene un bloque de cabeceras completo y no contiene ninguna trama Continuation
PADDED	Es el bit 0x8, cuando se activa, indica que los campos Pad Length y Padding están presentes

PING

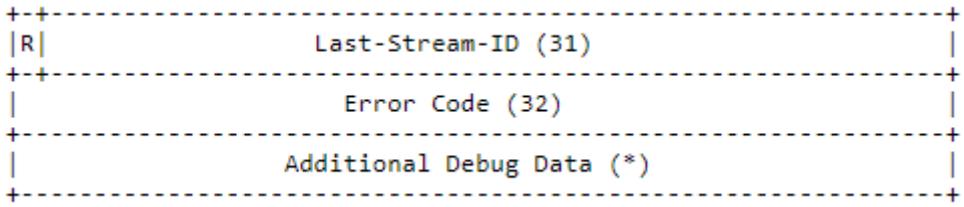
La trama Ping se utiliza para medir el tiempo mínimo entre los extremos y sirve también para saber si un extremo está operativo. Las tramas Ping contienen 8 octetos de datos opacos en el payload y estas tramas tienen que tener mayor prioridad que otra trama. Su valor en el campo Type es 0x6.



Flag	Descripción
ACK	Es el bit 0x1, cuando se activa, indica que la trama es una respuesta.

GOAWAY

La trama Goaway se utiliza para cerrar una conexión, permitiendo parar de aceptar nuevos flujos. Su valor en el campo Type es 0x7.

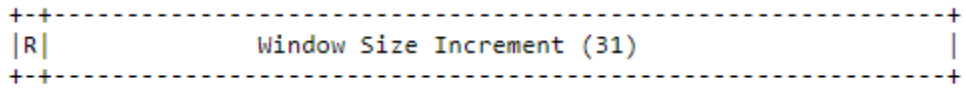


Campo	Descripción
R	Un bit reservado.
Last-Stream-ID	Indica el identificador más alto que ha recibido o procesado el emisor del GOAWAY con un entero de 31 bits
Error Code	Un valor de 32 bits sin signo que identifica el código de error
Additional Debug Data	Datos que puede enviar el emisor del GOAWAY para indicar más información acerca de los problemas.

Las tramas Goaway sirven para que el emisor pueda descartar tramas con un identificador superior al ultimo recibido y puede ser enviado múltiples veces si las circunstancias lo requieren. No tiene Flags este tipo de trama.

WINDOW_UPDATE

La trama Window_Update se utiliza para establecer un control de flujo. Su valor en el campo Type es 0x8.

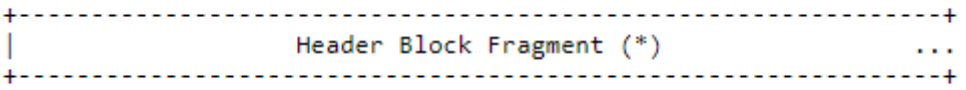


Campo	Descripción
R	Un bit reservado.
Window Size Increment	Un valor de 31 bits sin signo, que establece la cantidad de objetos puede transmitir además de los ya existentes. El rango máximo de incremento es de 1 a 2^{31} -1.

Las tramas Window_Update pueden especificarse a un flujo a la conexión entera, si el identificador es 0, es para toda la conexión. Este tipo de trama no tiene Flags.

CONTINUATION

La trama Continuation se utiliza para enviar más cabeceras que no han podido ser transmitidas anteriormente, la cantidad de tramas Continuation es ilimitada mientras sean del mismo flujo. Su valor en el campo Type es 0x9.



Campo	Descripción
Header Block Fragment	Son las cabeceras del mensaje.

Flag	Descripción
END_HEADERS	Es el bit 0x4, cuando se activa, indica que contiene un bloque de cabeceras completo y no contiene ninguna trama Continuation

Las tramas Continuation tienen que ser precedidas de una trama Headers, Push_Promise o Continuation sin el flag de END_HEADERS activado.

Anexo B Tutorial Webpagetest

Para realizar los test mediante la herramienta de Webpagetest es necesario tener en cuenta una serie de consideraciones que se exponen a continuación:

Una vez se accede a la url: https://www.webpagetest.org/

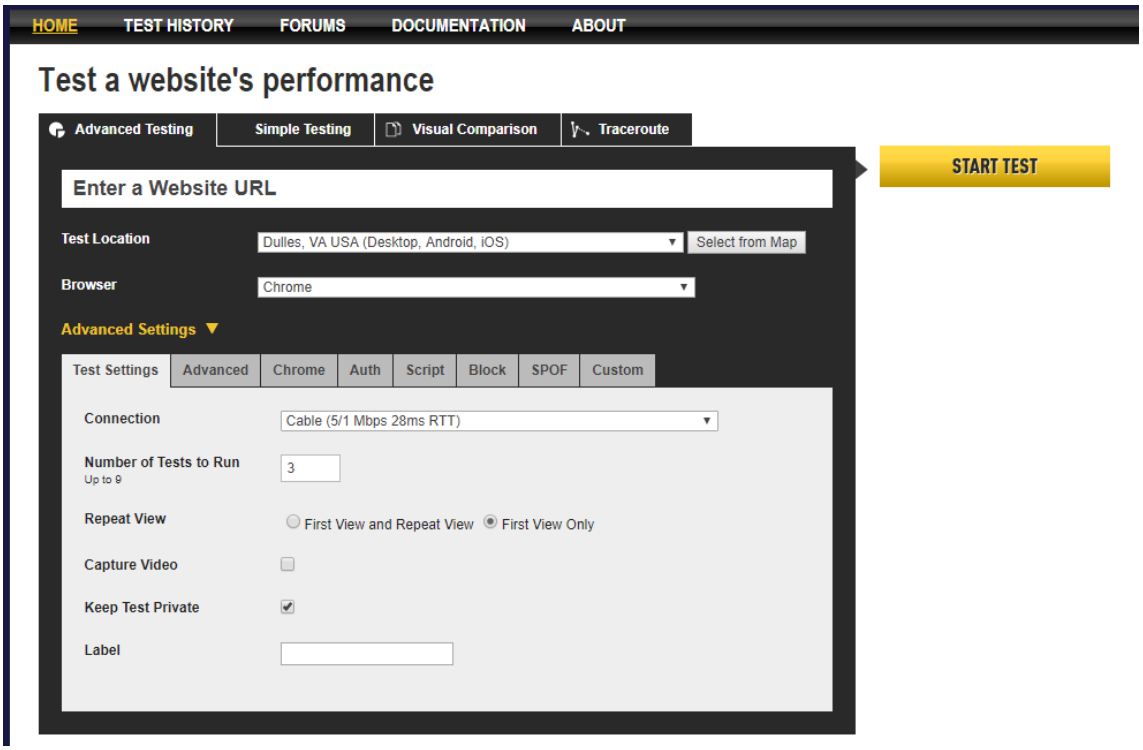


Figura 42 Tutorial Webpagetest 1/7

Hay que rellenar los siguientes apartados:

Enter a Website URL: Hay que especificar la url a la cual se desea acceder

Test Location: Se selecciona el sitio desde el cual se va a realizar el test, en el caso de este estudio se ha escogido London, UK -EC2 (Chrome, Firefox), puesto que es un servidor cercano a España para realizar el test y poder así realizar un test lo más parecido posible.

Browser: Se selecciona el navegador deseado, en este caso Chrome.

En Test Settings se pueden modificar los siguientes aspectos:

Connection: Se puede seleccionar una conexión estándar o personalizarla.

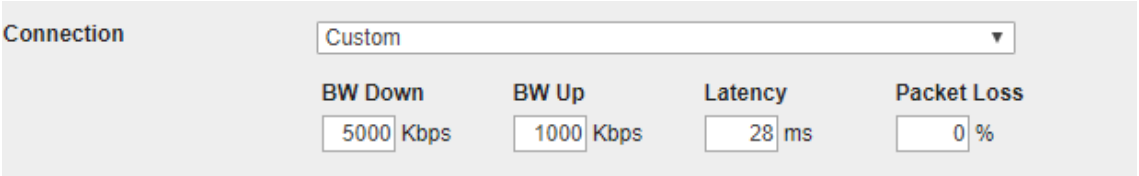


Figura 43 Tutorial Webpagetest 2/7

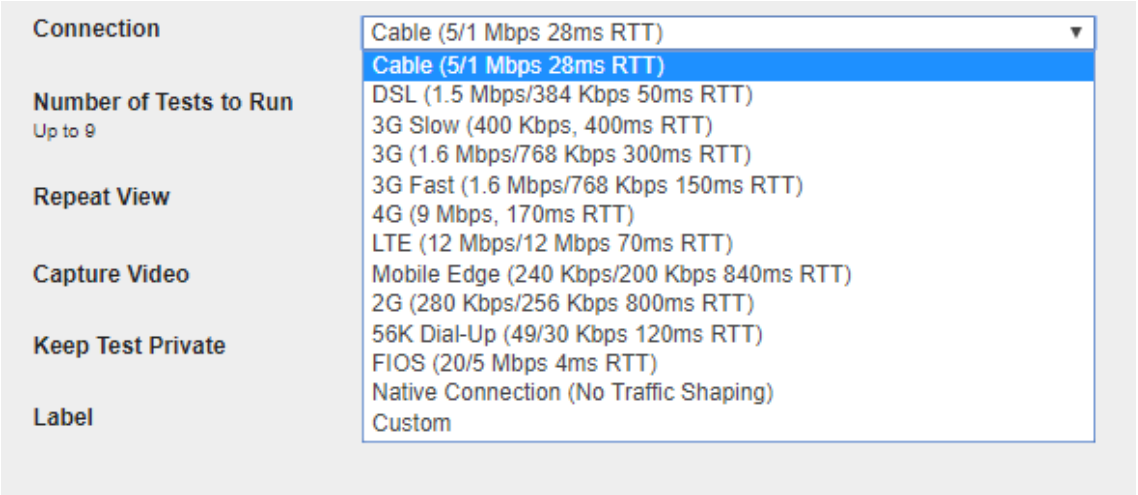


Figura 44 Tutorial Webpagetest 3/7

Number of Test to Run: Es importante establecer un número impar, para tener la mediana bien definida, lo recomendable es 9.

Repeat View: Hay que seleccionar si solo se quiere cargar una página por primera vez o si se desea que también cargue una vez está en cache, si se desea esto último hay que cambiar la opción. En este estudio se deja por defecto la opción de cargar solo por primera vez, por lo que cada test lo realizará sin tener en cuenta la cache.

Label: Es el nombre con el que quedará almacenado el test.

Otra opción interesante es la opción para desactivar HTTP/2 en Webpagetest. Para ello hay que seleccionar la opción de Chrome Command-line y poner –disable-http2

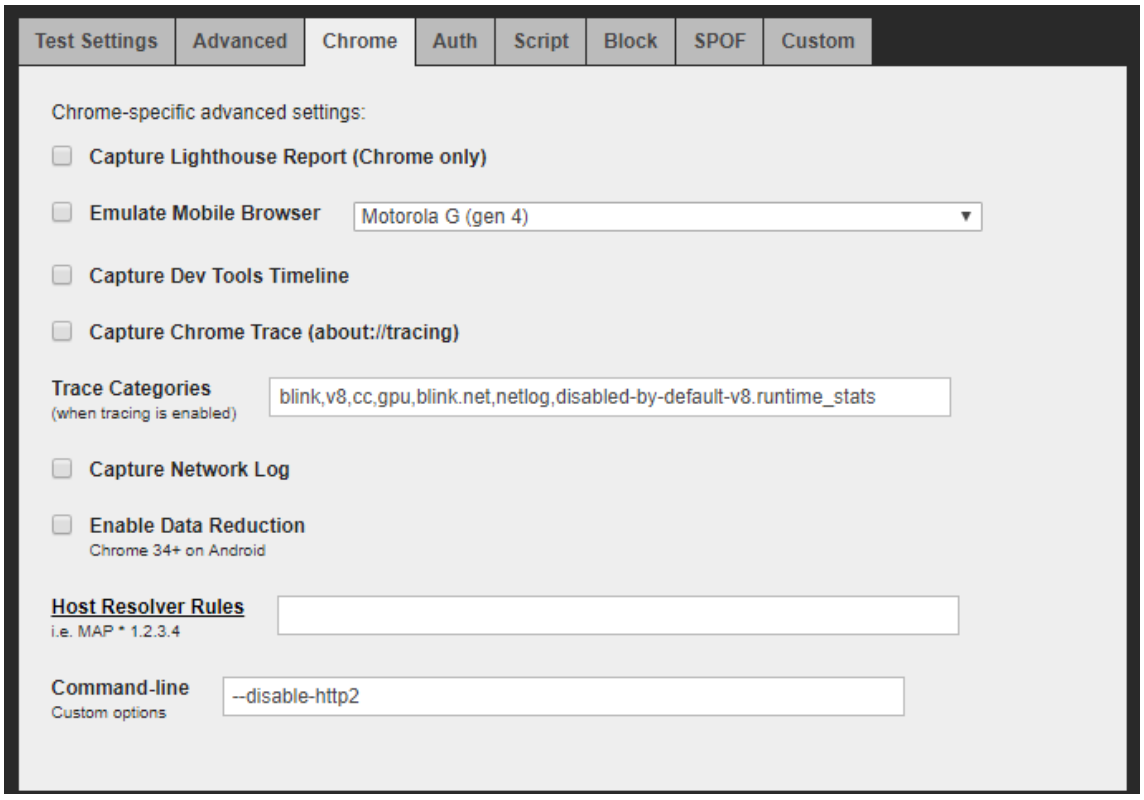


Figura 45 Tutorial Webpagetest 4/7

Para visualizar los Test realizados, hay que ir a la pestaña Test History, ahí se pueden elegir los días que han transcurrido desde el test, para filtrar los test.

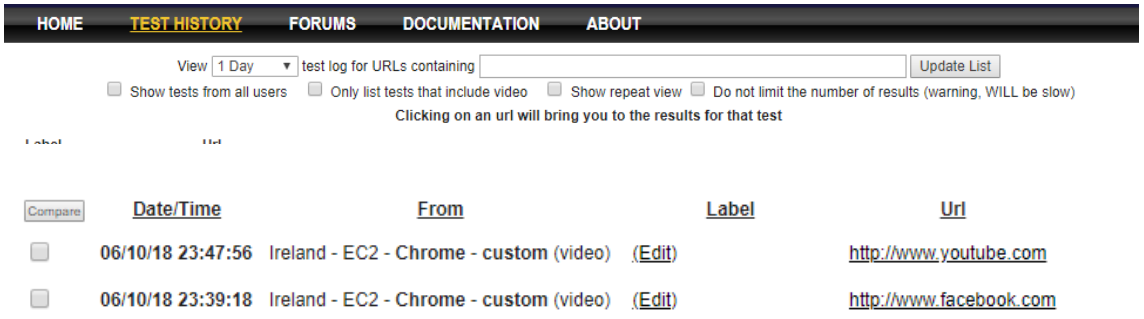


Figura 46 Tutorial Webpagetest 5/7

Y se pueden comparar varios test haciendo clic a la izquierda de cada test y luego haciendo clic en compare:

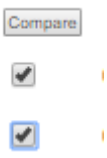


Figura 47 Tutorial Webpagetest 6/7

Y como resultado sale la siguiente imagen para comparar:

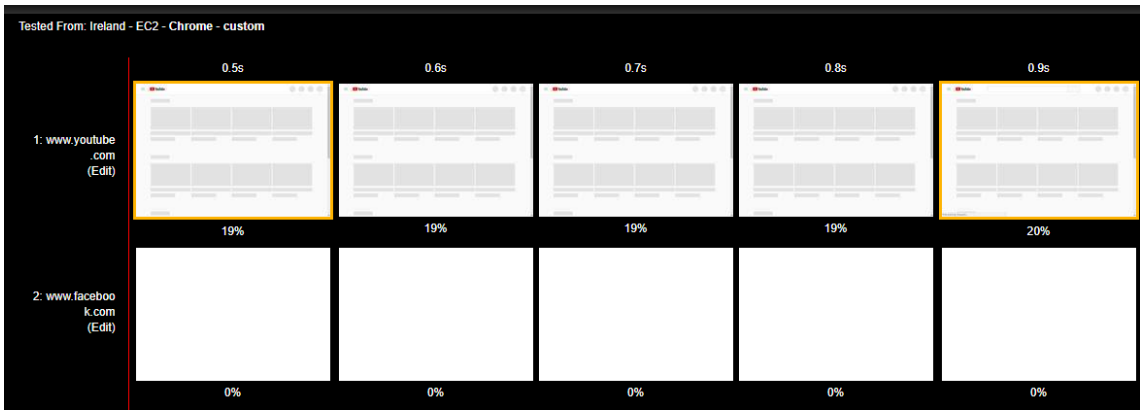


Figura 48 Tutorial Webpagetest 7/7

Además la comparación da como resultados el progreso visual o demás estadísticas:

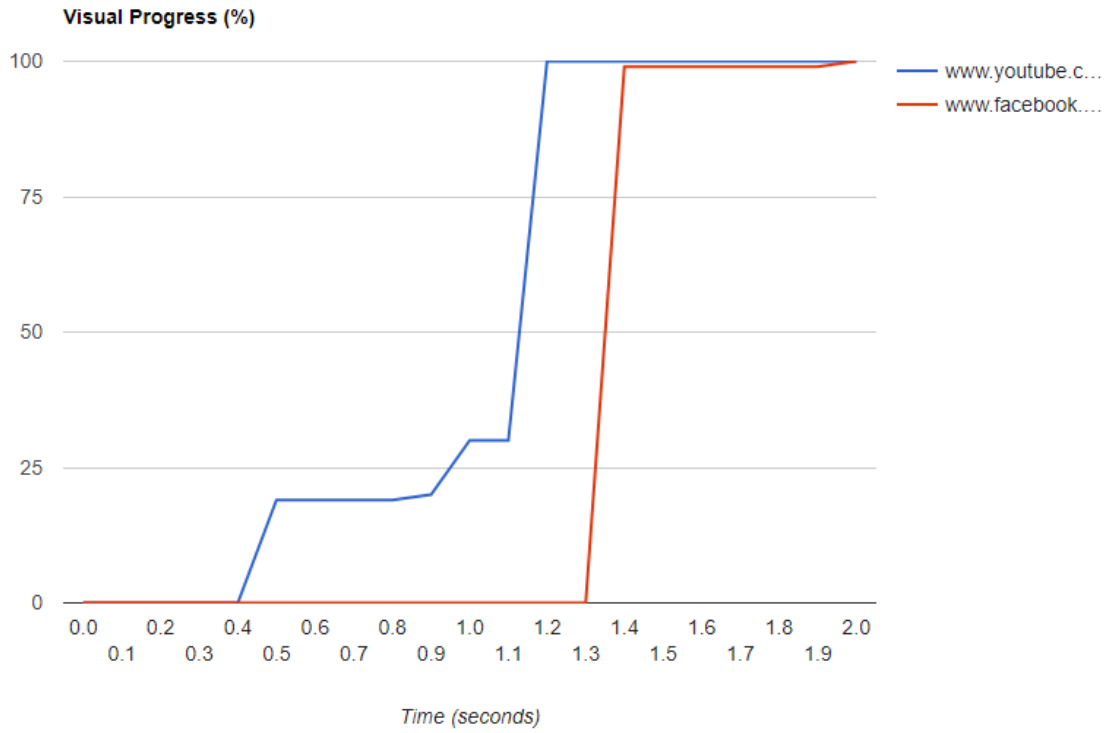


Figura 49 Estadísticas Webpagetest 1/2

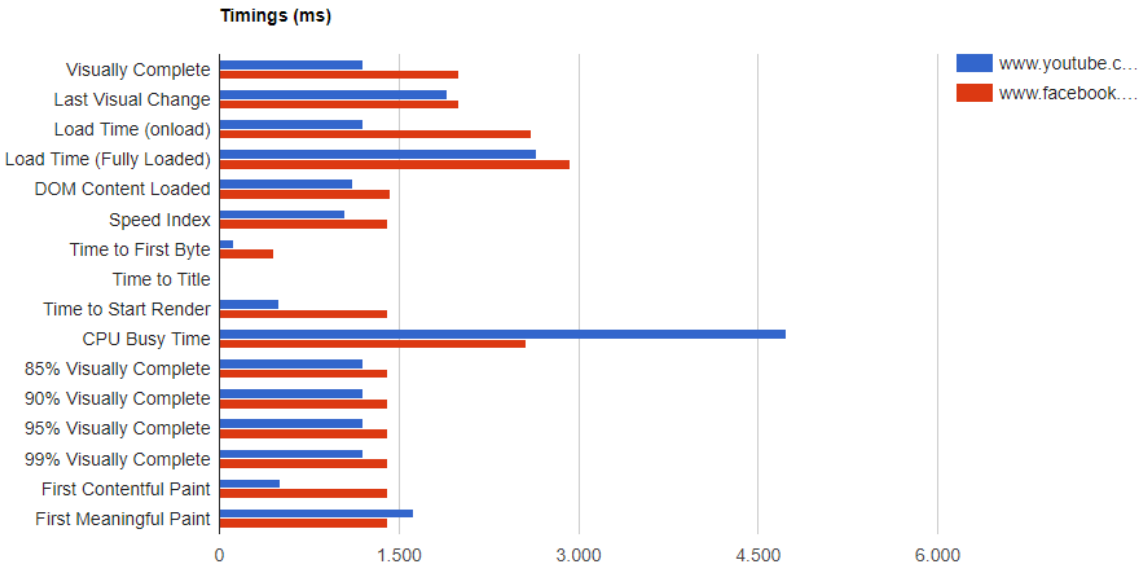


Figura 50 Estadísticas Webpagetest 2/2

Los test realizados son los siguientes:

ADSL

50 ms HTTP/2

https://www.webpagetest.org/result/180528_K4_02db3ac01cf1f34c4b060b0db07aa1bc/ - YouTube

https://www.webpagetest.org/result/180528_R3_6c2fdb7a5999b13c8982904c79075c0a/ - Twitter

https://www.webpagetest.org/result/180528_KX_df72ae902f7edc867f88d9e2c2766534/ - Facebook

50 ms HTTP/1.1

https://www.webpagetest.org/result/180528_SQ_ae6572b08d9b5df05d1998972956ec72/- YouTube

https://www.webpagetest.org/result/180528_1Q_174d8c00925ac8f54dd07a62615dfa8d/ - Twitter

https://www.webpagetest.org/result/180603_DP_669526d9ec1467123a668e9047492542/ - Facebook

100 ms HTTP/2

https://www.webpagetest.org/result/180528_CH_be029ac3606433d0c1e9a2815c6d8d48/ - YouTube

https://www.webpagetest.org/result/180528_3X_588cbeb0a987842247b1317c4f3a1039/- Twitter

https://www.webpagetest.org/result/180528_RP_a15cb4740b8b41c145c6d1bf5bf5acb5/ - Facebook

100 ms HTTP/1.1

https://www.webpagetest.org/result/180528_36_0fa32b98ab094d07ecce15dc5ce54df5/- YouTube

https://www.webpagetest.org/result/180528_W2_46b4781514021e0f70fc26a621dbca48/- Twitter

https://www.webpagetest.org/result/180528_PT_741e7493d3c3ac039e42a94f388687d3/- Twitter

150 ms HTTP/2

https://www.webpagetest.org/result/180528_VA_38eef7be6fe9301c05497b0328edd95c/- YouTube

https://www.webpagetest.org/result/180528_70_41e6013e8835b717445c62241b7c261e/- Twitter

https://www.webpagetest.org/result/180528_1R_a1f2112a29673e13bf88b071e557a934/- Facebook

150 ms HTTP/1.1

https://www.webpagetest.org/result/180528_F2_57230926bdf7f161ba4bd1d1e0796a91/- YouTube

https://www.webpagetest.org/result/180528_4M_b2822eb38803786b849ec03c75f161a7/- Twitter

https://www.webpagetest.org/result/180528_79_a2918c8f7fb486cdc058f415b394d7bf/- Facebook

3G

150 ms HTTP/2

https://www.webpagetest.org/result/180528_1S_531cb5601af793598aea555ea0900b3f/- YouTube

https://www.webpagetest.org/result/180528_JK_4cf10b6f93780428ed580f5b3d70fb69/- Twitter

https://www.webpagetest.org/result/180528_MR_9a1f33b9d1375f4db6181c470ea5eae3/- Facebook

150 ms HTTP/1.1

https://www.webpagetest.org/result/180528_57_28a9c47aefa987a6ef0a923391468bbf/- YouTube

https://www.webpagetest.org/result/180528_BY_d030da00d271b1f2f0c9fcf1597df3f9/- Twitter

https://www.webpagetest.org/result/180528_TX_44cf029b4e85c579875afb5d2061a3d0/- Facebook

300 ms HTTP/2

https://www.webpagetest.org/result/180528_HE_c5aa715f32187787f83d0c1edab5e16b/- YouTube

https://www.webpagetest.org/result/180528_PF_21f30ae5d5157450debeebbce4b0364d/- Twitter

https://www.webpagetest.org/result/180528_R1_19117eab0e8867f8e70609824527f3df/- Facebook

300 ms HTTP/1.1

https://www.webpagetest.org/result/180528_FH_ef85b97e30bad8a53cbc8847d1ab3d92/- YouTube

https://www.webpagetest.org/result/180528_94_5197955f4169fce05d1bc27a549e317b/- Twitter

https://www.webpagetest.org/result/180528_HB_9456862d06106835b62379822056ede0/- Facebook

400 ms HTTP/2

https://www.webpagetest.org/result/180528_B8_d61a649c1013f527416253633586c1e4/- YouTube

https://www.webpagetest.org/result/180528_SD_66bfb2a0a38339e932a8a48b4b781ad2/- Twitter

https://www.webpagetest.org/result/180528_9N_b6f0f4c0b590d3c5c2634dbed97b8c4a/- Facebook

400 ms HTTP/1.1

https://www.webpagetest.org/result/180528_35_2fd9c8cc665cbc009b424f4f923ecf4c/- YouTube

https://www.webpagetest.org/result/180528_8M_1543a33e82d824498e297d12360100c8/- Twitter

https://www.webpagetest.org/result/180528_EB_0c208fb59645e54a08060fcef222f459/- Facebook

4G

70 ms HTTP/2

https://www.webpagetest.org/result/180529_NE_06d3f7ed6a876c57423c0b4b3f903075/- YouTube

https://www.webpagetest.org/result/180529_CQ_2fe3660576c78fd022112a62c37a57c6/- Twitter

https://www.webpagetest.org/result/180529_A6_6fad8dfc102bab5538097c032fe395f7/- Facebook

70 ms HTTP/1.1

https://www.webpagetest.org/result/180529_3F_1084382c41a3b9041e5645ccb8f3f269/- YouTube

https://www.webpagetest.org/result/180529_EY_649c3ce9bd06fe78dfd9ae4d35781802/- Twitter

https://www.webpagetest.org/result/180529_P9_e318d9b58553b7713679707e09f7106f/- Facebook

120 ms HTTP/2

https://www.webpagetest.org/result/180529_HQ_4085a90302a83460dd9d025df6cba62c/- YouTube

https://www.webpagetest.org/result/180529_YK_c651085bd27566c6289451ae383f5bcb/- Twitter

https://www.webpagetest.org/result/180529_7C_9ffba4e7815967fc2073a6d2029d3435/- Facebook

120 ms HTTP/1.1

https://www.webpagetest.org/result/180529_2Z_ade352a90da17d6d570bc99c9d70c0f8/- YouTube

https://www.webpagetest.org/result/180529_P4_f9fe887e28be6c3ee6d9747a7b5f2ad7/- Twitter

https://www.webpagetest.org/result/180529_B0_f77afb844b1fa030f5322bcb366b0b9f/- Facebook

170 ms HTTP/2

https://www.webpagetest.org/result/180529_3N_cbf23d2665e1ef71aa378c7895f9110b/- YouTube

https://www.webpagetest.org/result/180529_VR_e23478364d078e07e1c5d47ba4ea7721/- Twitter

https://www.webpagetest.org/result/180529_9Z_649e39e5a3dd3f94fd26e89335da93ef/- Facebook

170 ms HTTP/1.1

https://www.webpagetest.org/result/180529_52_dfc8b45e5ce5b42ef6f813ff0ab0873f/- YouTube

https://www.webpagetest.org/result/180529_10_197c16a757eb41501506c7506e3708e2/- Twitter

https://www.webpagetest.org/result/180529_52_dfc8b45e5ce5b42ef6f813ff0ab0873f/- Facebook

Fibra Óptica

20 ms HTTP/2

https://www.webpagetest.org/result/180529_A5_6ceef7a0adf778904a819e58ce3d2039/- YouTube

https://www.webpagetest.org/result/180529_DS_510e926c6e8da75c4dda5f2fec7005d8/- Twitter

https://www.webpagetest.org/result/180529_SE_e07e7421c2b7b559a89f8a5f0215b154/- Facebook

20 ms HTTP/1.1

https://www.webpagetest.org/result/180529_EZ_5951dec3e9d2512fa7aaf7ec79d1d9a8/- YouTube

https://www.webpagetest.org/result/180529_7H_6b384a4653f33dd11135d0791966c671/- Twitter

https://www.webpagetest.org/result/180529_AD_450410d8828eaba8ac7eedb6cc7236cc/- Facebook

50 ms HTTP/2

https://www.webpagetest.org/result/180529_RB_ae94ae8c1c00a6516a0479a1015c1c17/- YouTube

https://www.webpagetest.org/result/180529_7R_856df6e8ad12536255d3680e9e46f8bb/- Twitter

https://www.webpagetest.org/result/180529_5R_b0759d5a06d5b3590a5bf28c30f45b5a/- Facebook

50 ms HTTP/1.1

https://www.webpagetest.org/result/180529_WZ_ca617543e8051f58caf79c6350077cec/- YouTube

https://www.webpagetest.org/result/180529_MQ_17d7d1cc00814af108342a4d1c3618e3/- Twitter

https://www.webpagetest.org/result/180529_5M_0ba63be80c05303f886a476e80fcf1d5/- Facebook

80 ms HTTP/2

https://www.webpagetest.org/result/180529_SH_543bcf7c6d0dab82948eba63962bbe13/- YouTube

https://www.webpagetest.org/result/180529_G3_6ed294d5308e53d07142eac03aa5bcc0/- Twitter

https://www.webpagetest.org/result/180529_2H_93b0cfb75f5d163cbc86f5398186aa5c/- Facebook

80 ms HTTP/1.1

https://www.webpagetest.org/result/180529_2T_13a1cc5cbe916a99d4e5f2d824383113/- YouTube

https://www.webpagetest.org/result/180529_GD_380a1fa5238ec480bd177dca05f23dae/- Twitter

https://www.webpagetest.org/result/180529_GD_380a1fa5238ec480bd177dca05f23dae/- Facebook

Anexo C Analizar Trafico HTTP/2

Para analizar el tráfico HTTP/2 [14] se puede capturar mediante Wireshark, que es un analizador de red gratuito, para ello es necesario agregar la siguiente línea en las propiedades de Google Chrome en la Opción destino para registrar las claves SSL

--ssl-key-log-file=%USERPROFILE%\sslkeysSSLKey.pms

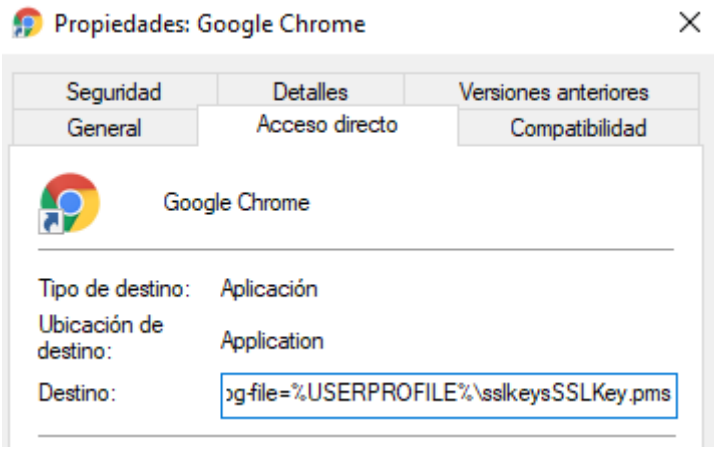


Figura 51 Registrar SSL en Chrome

Ahora en Wireshark se abre Preferencias -> Protocolos -> SSL

Y en Pre-Máster Secret log filename se le pondría la ruta del registro de las claves ssl

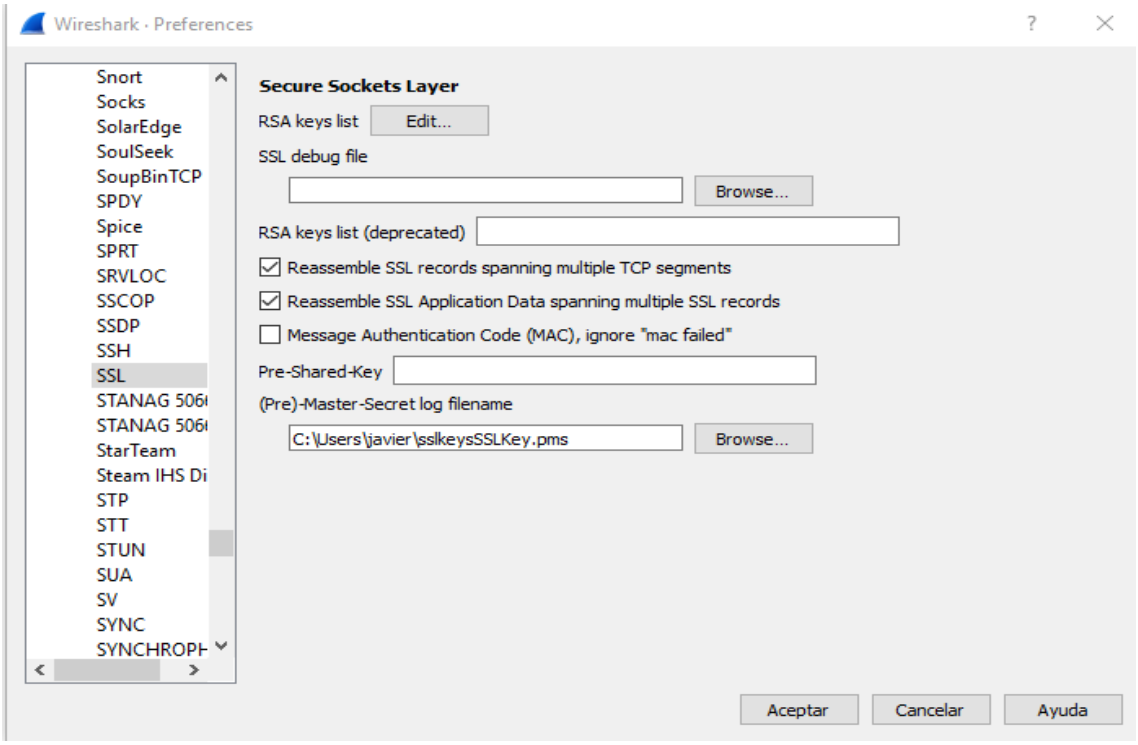


Figura 52 Agregar las claves SSL en Wireshark

Una vez se ha cargado el fichero, se puede observar el protocolo al completo:

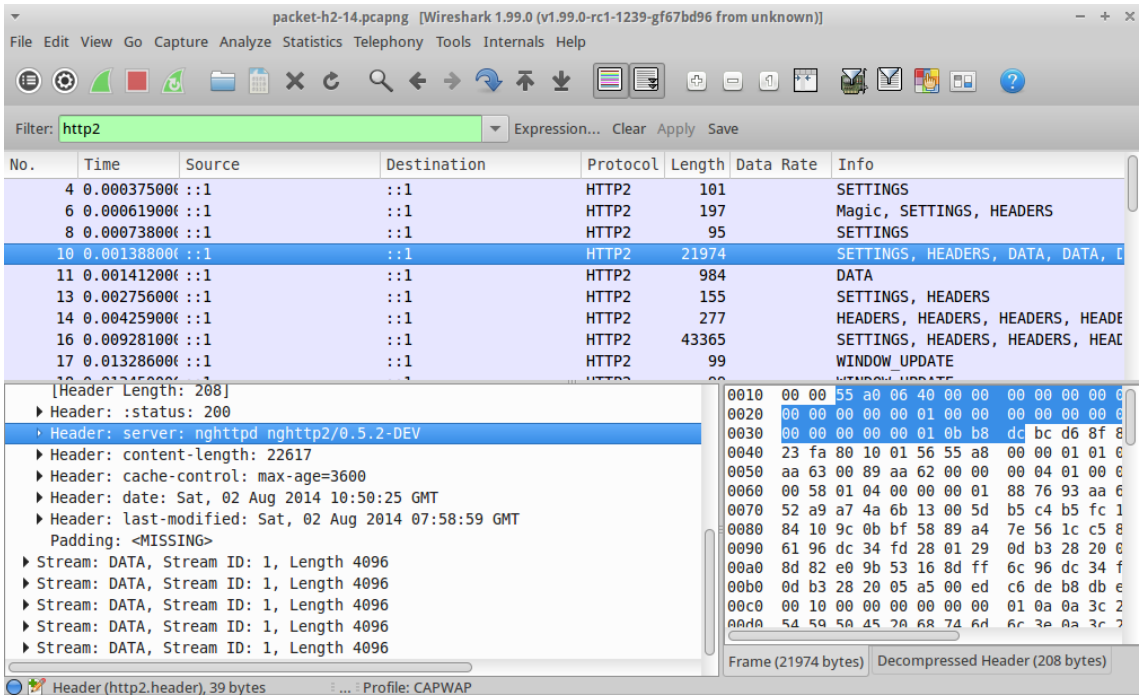


Figura 53 HTTP/2 en Wireshark

Otra opción para ver las tramas es acceder a la url de Chrome:

chrome://net-internals/#http2

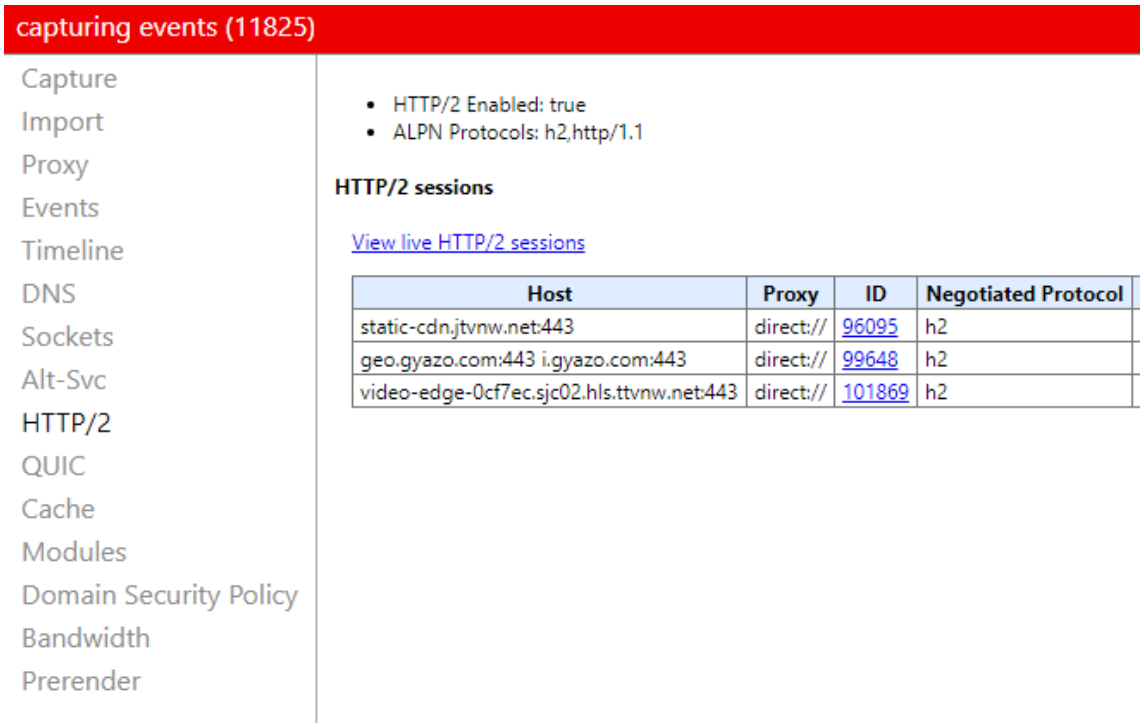


Figura 54 Sesiones HTTP/2 Chrome

que nos muestra las sesiones de HTTP/2, una vez se hace clic en View live HTTP/2 sessions accedemos a las sesiones activas de HTTP/2

Figura 55 Sesiones Activas HTTP/2

En este caso vemos que hay 2 activas y si clicamos en una de ellas, podemos ver los paquetes:

Figura 56 Tramas HTTP/2 en Chrome

Para visualizar estos datos en Android también es posible accediendo a la misma URL, que nos muestra la siguiente pantalla

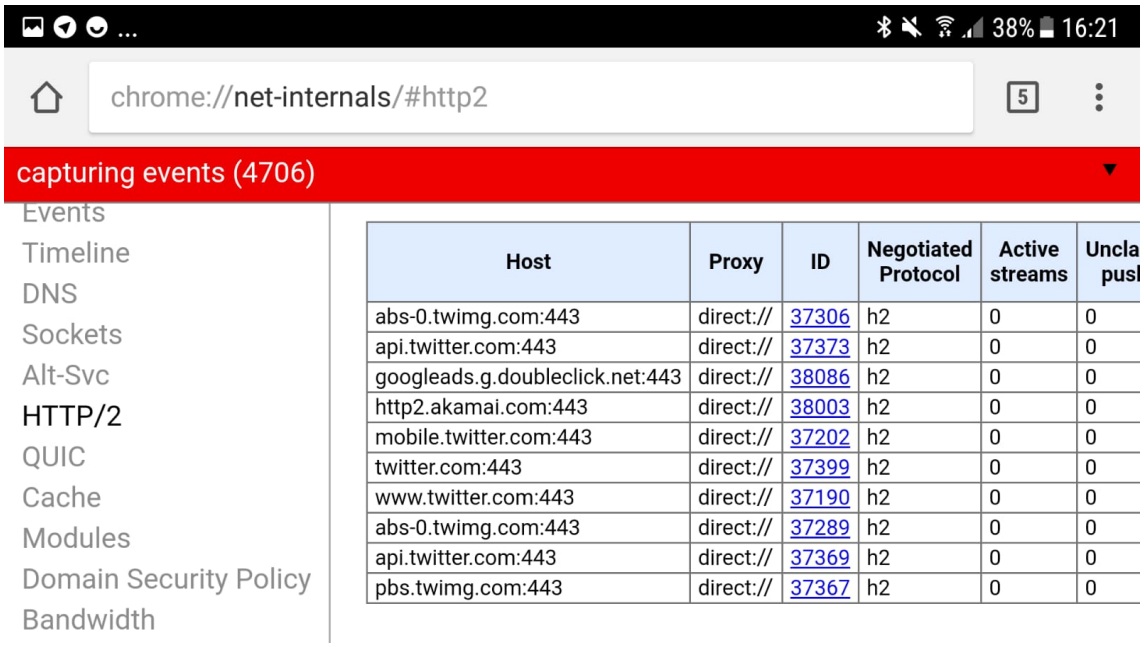


Figura 57 Sesiones HTTP/2 en Android 1/2

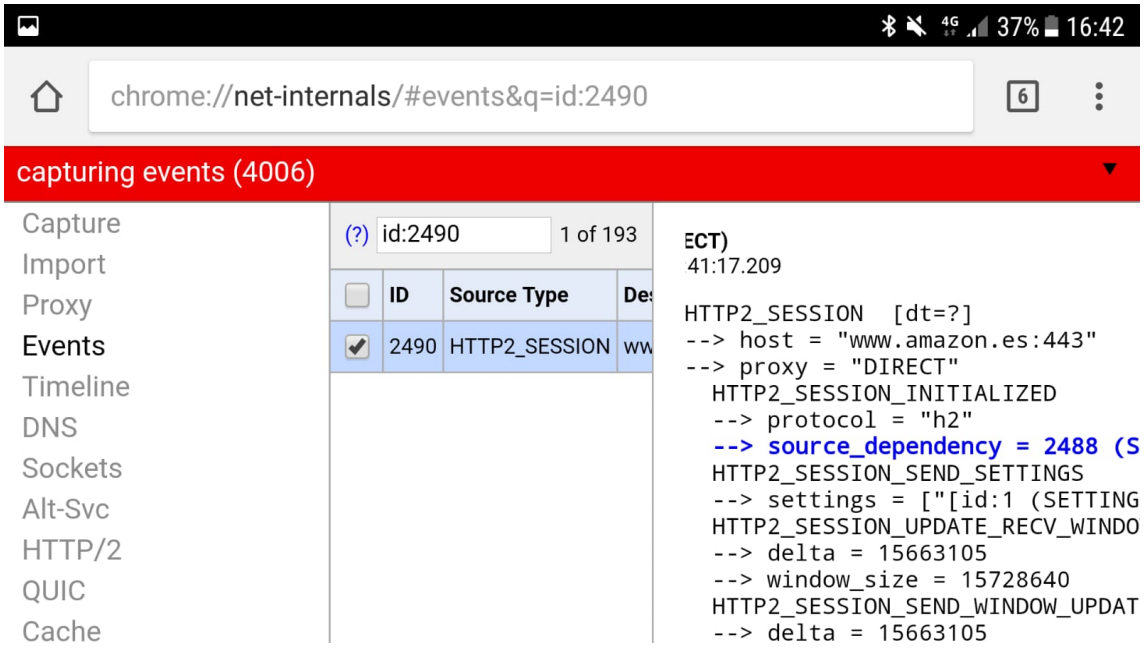


Figura 58 Sesiones HTTP/2 en Android 2/2

Si queremos analizar los tiempos de carga en Android [13], hay que realizar los pasos siguientes para poder analizar desde la herramienta de desarrolladores de nuestro pc.

El primer paso es acceder a la información de nuestro software

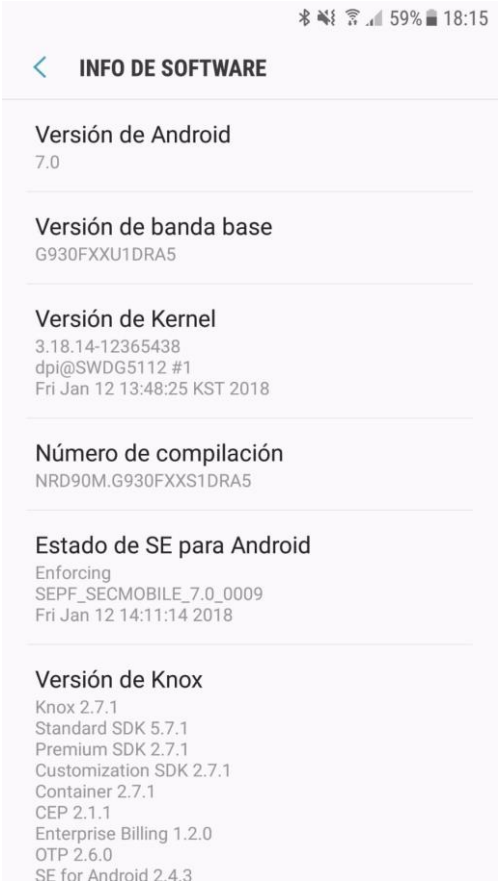


Figura 59 Guía Analizar Android 1/5

A continuación, hay que presionar el número de compilación durante unos segundos para acceder al modo desarrollador:

Figura 60 Guía Analizar Android 2/5

que aparecerá como una de las ultimas opciones en ajustes.

Una vez entramos tendremos que activar la opción depuración de USB

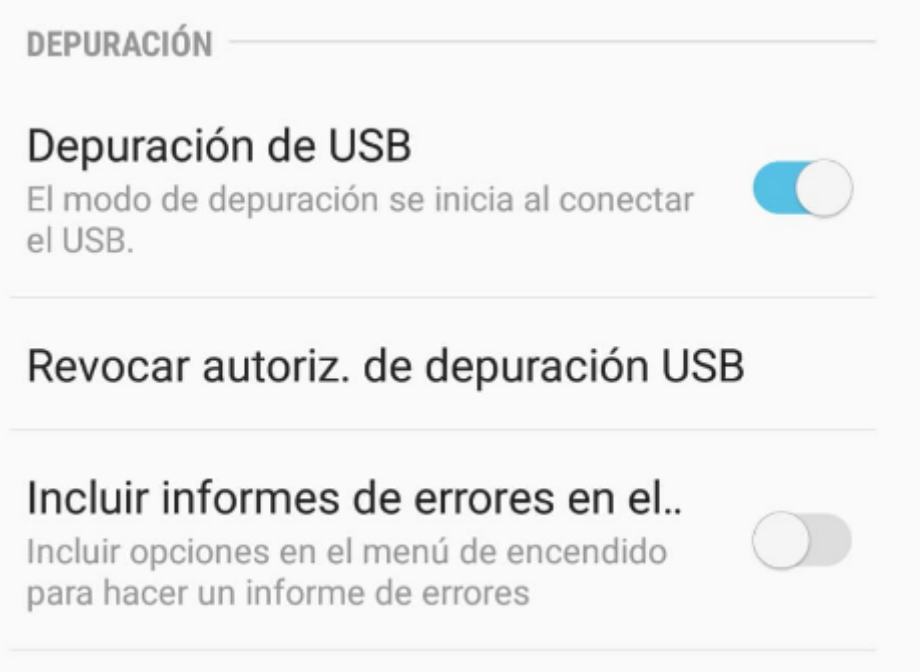


Figura 61 Guía Analizar Android 3/5

A continuación, conectamos con USB nuestro móvil al pc y volvemos a nuestro pc para acceder a la herramienta para desarrolladores de pc, comprobamos que la opción de descubrir dispositivos USB.

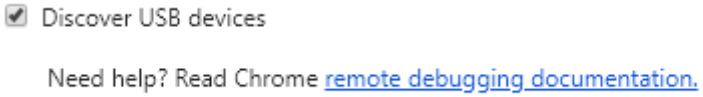


Figura 62 Guía Analizar Android 4/5

A la izquierda nos mostrará que está conectado:

Figura 63 Guía Analizar Android 5/5

Y al hacer clic en el dispositivo podremos navegar como si fuera nuestro pc y poder analizarlo.

Anexo D Pasos para instalar un servidor HTTP/2

Para implantar un servidor en HTTP/2 se necesitan hacer 2 pasos, el primero de ellos es instalar un servidor web que interactúe con HTTP/2. El segundo es obtener un certificado TLS para que el navegador interactúe con el servidor.

Puesto que los navegadores web soportan HTTP/2 con cifrado TLS, se procede a obtener un certificado propio. Hay varios métodos que pueden ser utilizados, uno de ellos es usar un generador de certificados, otro es usar un certificado firmado por nosotros mismos y el ultimo es un certificado que está firmado por una Autoridad Certificadora (CA). Puesto que las dos primeras opciones generan avisos en los navegadores porque no están firmados por las CA pero pueden resultar utiles para probar que efectivamente se ha generado un certificado TLS, se usará una CA para que se genere el certificado.

Para ello se usa Let's Encrypts, que es una nueva CA, cuyo objetivo es generar un certificado fácilmente, automático y gratis para todo el mundo. Hay muchos clientes y librerías para elegir, pero el cliente más recomendado es el denominado certbot, puesto que debida a su simplicidad para obtener certificados, ser gratuita y recomendada por el libro [1] de O'Reilly.

Para obtener certbot en Linux las instrucciones son las siguientes:

wget https://dl.eff.org/certbot-auto
chmod a+x certbot-auto
./certbot-auto certonly –webroot -w <raíz de la web> -d <tu dominio>

(Sustituyendo la raíz de la web y el dominio por los valores reales.)

El nuevo certificado junto con la clave privada se encontrará en la carpeta:

/etc/letsencrypt/live/<tu dominio>/privkey.pem -> Tu clave privada

/etc/letsencrypt/live/<tu dominio>/cert.pem -> Tu nuevo certificado

Una vez realizado este paso, se procede a instalar un servidor que "hable" HTTP/2, hay múltiples opciones como apache, nginx, h2o, jetty…

Pero por simplicidad y rapidez se va a seleccionar nghttp2, diseñado por Tatsuhiro Tsujikawa que proporciona una cantidad de herramientas útiles para trabajar con HTTP/2.

En Ubuntu 16 para ponerlo en marcha se tendría que instalar y ejecutar mediante los siguientes comandos:

Sudo apt-get install nghttp2
./nghttpd -v -d <raíz de la web> <puerto> <clave privada> <certificado>

Sustituyendo la raíz de la web por la ruta de la web, el puerto por el puerto que se desea que escuche el servidor, clave privada por la ruta donde se ha generado la clave privada y el certificado por la ruta donde se ha generado el certificado anteriormente, se instala el servidor en HTTP/2.

Para finalizar, tan solo hay que descargarse un navegador web compatible con HTTP/2 y acceder a la página web, que se mostrará en HTTP/2.