Estudio del protocolo HTTP/2 usando la plataforma NodeJS

Diego J BOTIA
Ingeniería de Sistemas, Universidad de Antioquia
Medellín, Antioquia, Colombia

Santiago CADAVID
Ingeniería de Sistemas, Universidad de Antioquia
Medellín, Antioquia, Colombia

Juan F GIRALDO
Ingeniería de Sistemas, Universidad de Antioquia
Medellín, Antioquia, Colombia

RESUMEN

El protocolo *HTTP/2* presenta un conjunto de ventajas y características que permiten resolver los problemas que el protocolo *HTTP/1.1* trae desde su diseño, donde durante casi 15 años no se han hecho los cambios correspondientes para mejorar y adaptarse con las nuevas tecnologías que han venido apareciendo en Internet. Sin embargo, el protocolo aún es relativamente nuevo y desconocido, dejando así algunas dudas sobre sus nuevas características y formas de implementación. En este trabajo se adopta un enfoque metodológico que permite la implementación del protocolo *HTTP/2* en un servidor NodeJS usando el framework Express y la integración del módulo *SPDY*; todo esto bajo el módulo de pruebas de la aplicación desarrollada para este estudio *GTM²* (Gift-Trade-Meet & More). Además, se presenta un escenario de pruebas para hacer evaluaciones y posteriormente obtener resultados usando características como la multiplexación y mejora en los tiempos de carga brindadas por el protocolo; mediante un conjunto de imágenes en el servidor configurado con *HTTP/1.1* y *HTTP/2,* y así poder evaluar los tiempos promedio de carga y determinar en qué casos es mejor usar cada protocolo.

Palabras clave: HTTP/2, multiplexación, tiempos de carga, petición, SPDY, NodeJS, performance.

I. INTRODUCCIÓN

El protocolo *HTTP/1.1* se ha estado implementando en los navegadores y servidores desde 1999 y ha sido uno de los protocolos de comunicación más utilizados en Internet [1]. Sin embargo, con la evolución y desarrollo de aplicaciones cada vez más complejas y el crecimiento del tráfico y usuarios en Internet, se fueron acumulando a lo largo del tiempo varios problemas como:

- Redundancia en las peticiones (*User-agent problem*): ocurre porque el protocolo no guarda el estado de la sesión y siempre debe estar enviando las mismas cabeceras una y otra vez, malgastando ancho de banda [2].
- *Head of line blocking (HOL o HOLB):* afecta directamente a la conexión *TCP* porque bloquea el envío de solicitudes desde el navegador afectando con esto los tiempos de carga de los recursos web [2].
- Única conexión *TCP* por recurso: debido a su diseño, el protocolo *HTTP/1.1* realiza siempre una conexión por recurso solicitado para facilitar la carga de estos, sin embargo, esta acción representa un defecto porque aumenta el uso de ancho de banda y en general de recursos de red [2].
- El protocolo *HTTP/1.1* tiene ciertas brechas de seguridad que han causado molestias tanto a los desarrolladores como a los usuarios. Por ejemplo, no trae por defecto el soporte a *TLS/SSL* que subsanan ciertas vulnerabilidades, donde se cifra la información y existe autenticación entre aplicaciones y servidores. Cabe aclarar que a pesar de la existencia de los protocolos de seguridad actuales se siguen teniendo vulnerabilidades tales como: *POODLE* (*Padding Oracle On Downgraded Legacy Encryption*) y *DROWN (Decrypting RSA with Obsolete and Weakened eNcryption)* [3], los cuales buscan vulnerar al servidor por medio de los protocolos de transporte antes mencionados. Gracias a la implementación de *TLS* junto con *ALPN (Application-Layer Protocol Negotiation*) [4] se puede mitigar este problema.

Por lo descrito anteriormente y con el objetivo de mejorar el desarrollo de nuevas plataformas, la adopción de la computación en la nube, el aumento del tamaño y la cantidad de información que se transfiere desde el servidor al lado del cliente, hace que sea imprescindible el uso de un nuevo protocolo que mejore la comunicación y brinde mayores garantías. En 2009, Google decidió emprender un proyecto experimental sobre la creación de un nuevo protocolo llamado *SPDY* [5], el cual mantuvo la semántica de *HTTP* pero resolviendo el problema de *head of line blocking* que presenta el protocolo HTTP/1.1. Luego en 2012, Google abandona el proyecto debido a que se propuso el nuevo protocolo estándar *HTTP/2*, el cual fue lanzado por la *IETF* (*Internet Engineering Task Force*) mediante el RFC 7540 [3] como una versión preliminar basado en el protocolo *SPDY*, que fue evolucionando y presenta grandes ventajas como la compresión de cabeceras, multiplexación del canal de comunicación TCP, la característica de server *PUSH* que permite responder a las peticiones del cliente sin ni siquiera estas ser enviadas y guardar la respuesta en una caché local; capa de seguridad *TLS/SSL* implementada, entre otras. La Figura 1 presenta la línea de tiempo que muestra la evolución del protocolo y el largo periodo de tiempo (entre 1999 y 2009) donde no hubo un avance ni se hicieron correcciones a los problemas que este presentaba, mientras que

aparecían nuevas tecnologías para la web como AJAX, servicios web SOAP/ RESTful, Websockets, entre otros que hacían necesario que el protocolo HTTP/1.1 evolucionará.

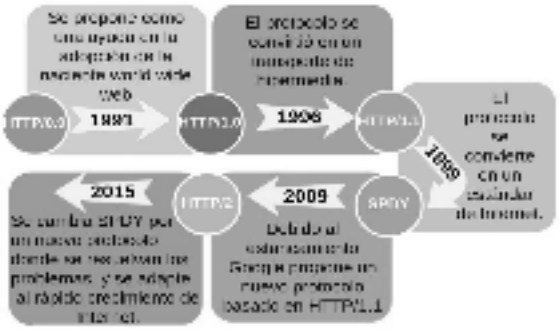

Figura 1. Línea del tiempo del protocolo *HTTP. Fuente: Los Autores.*

El protocolo *HTTP/2* ayuda a la atención de varias peticiones a la vez, asegura el canal de comunicación, y en definitiva mejora el tiempo de respuesta hacia el usuario. La Figura 2 representa la multiplexación del canal *TCP* y cómo son servidas las peticiones por cada uno de los protocolos, donde se evidencia que *HTTP/2* tiene la característica de servir todos los recursos en una sola conexión gracias a la multiplexación de conexiones.

bosque

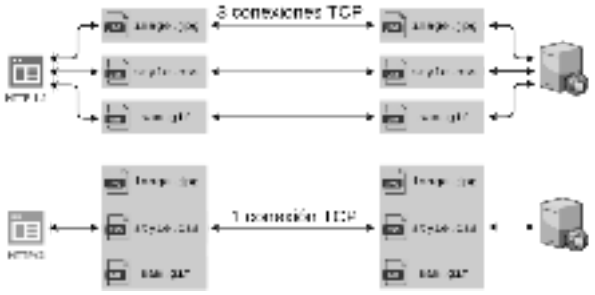

Figura 2. Multiplexación del canal *TCP* en el protocolo *HTTP/1.1* y *HTTP/2. Fuente: Los Autores.*

En este trabajo se analizan algunas de las características anteriormente mencionadas y se realizó pruebas sobre un módulo específico de *test* de la aplicación *GTM*[2] (*Gift-Trade-Meet & More*), usando el protocolo *HTTP/1.1* y *HTTP/2* en un servidor NodeJS, además se realizan pruebas concretas sobre varias imágenes en formato *JPEG* de diferente tamaño, y de esta manera hacer un análisis comparativo entre ambos protocolos.

El artículo está organizado de la siguiente manera: la sección II presenta el estado del arte que sirvió como base para este trabajo, la sección III presenta la metodología empleada, la sección IV describe los escenarios de prueba y la implementación del protocolo *HTTP/2* en el servidor NodeJS, la sección V detalla los resultados obtenidos con su respectivo análisis, y por último la sección VI presenta las conclusiones y el trabajo futuro pendiente por realizar.

II. ESTADO DEL ARTE

De Saxcé, H., et. al [2], presenta un trabajo de investigación e implementación que tiene como objetivo determinar el rendimiento entre los protocolos *HTTP/2* y *HTTP/1.1*, y qué ganancia obtienen los servidores que migran a *HTTP/2*. Para llegar a dicho objetivo realizaron pruebas sobre las características del protocolo *HTTP/2* como el "*server push*". Como criterio de evaluación se definió el tiempo de carga de la página, ya que consideran la métrica más importante para usuario final y el proveedor de contenido. En este trabajo los autores removieron la capa de seguridad para no sesgar las pruebas que se realizaron, y por lo tanto, usaron el navegador Chromium con soporte de *HTTP/2*, el cual permite desactivar la capa de seguridad (*TLS*). Bajo este mismo concepto decidieron emplear a H2O como servidor de aplicaciones principal. En cuanto a contenido se usaron archivos *HTML*, *CSS, JS* e imágenes para hacer las pruebas y dicho escenario se llevó a cabo en una red local *(LAN)* a través de una conexión de Internet banda ancha. Los resultados que se obtuvieron son favorables para *HTTP/2* visto de un modo general, donde los tiempos de carga para cantidades de imágenes desde 1 hasta 100, se conservan aproximadamente constantes para *HTTP/2*, mientras que para *HTTP/1.1* incrementa linealmente, de allí que se observe el problema de bloqueo *HOLB* para este último. Otro punto que destacan en el desarrollo de las pruebas, es el hecho de que en páginas como motores de búsqueda, el protocolo *HTTP/2* no muestra mucho efecto o mejoría, sin embargo, en páginas con alto contenido visual como imágenes se observa un gran beneficio. En general, el artículo describe una mejoría en los tiempos de carga y en la influencia de nuevas características para dar mejor desempeño en la web por parte del protocolo *HTTP/2*.

Por otro lado, Zarifis, K., et. al [6] muestra una investigación que busca principalmente la detección de tiempos de carga de página *(PLT - Page Load Time)* en 55 sitios conformados con un promedio de 111 objetos, los cuales hacen peticiones a servidores *Akamai* configurados en *HTTP/1.1* y *HTTP/2*, donde se encontró que el 60% de las veces *HTTP/2* tiene un *PLT* más pequeño, el 28% es insignificante y hay otros sitios en los que conduce a una mayor degradación del rendimiento. Dentro de sus pruebas también incluyen un modelo *RT-H2* el cual toma los datos de sincronización de página y estima la diferencia en tiempos de carga para *HTTP/1.1* y *HTTP/2.* Para hacer esto, *RT-H2* modela cuatro componentes importantes de *HTTP/2*: multiplexación, server push, priorización e intercalado de marcos.

Para validar los resultados obtenidos en las pruebas, realizaron una comparación de los resultados entre el modelo *RT-H2* y las mediciones reales de *PLT* que se habían hecho inicialmente. Finalmente se concluye que en el 90%, las características básicas de *HTTP/2* representan una mejoría para casi dos tercios de los sitios de la web, donde características como server push y priorización son las que más influyen, pero esto no significa que el protocolo esté terminado, aún hay más características en las

que debe mejorar.

III. METODOLOGÍA

Para la implementación y el desarrollo de la aplicación *GTM²* integrando el protocolo *HTTP/2*, el análisis de resultados y el despliegue de la aplicación en los dos protocolos, se hizo uso del marco de trabajo SCRUM [7] que hace parte de las metodologías ágiles de desarrollo de software y así se realizó la definición y priorización de historias de usuario, la definición de los sprints semanales, y la retroalimentación por parte del product owner.

Para realizar las pruebas de carga y evaluación de las métricas, se definió un escenario de pruebas que se describe en la siguiente sección de manera más detallada y en el cual se utilizó como principal criterio el tiempo de carga de imágenes que es otorgado por la sección de red en las herramientas de desarrollador que Chrome (navegador de Google) nos brinda. Para cada petición se muestra de manera detallada y gráfica el tiempo que tomó la petición en ser enviada, *TTFB (time to first byte)*, el tiempo en cola, etc. Principalmente se va a tener en cuenta el tiempo total que es la suma de todos y su respectivo promedio. El estudio es llevado a cabo sobre la aplicación que se describe en la siguiente sección donde se cargan dos conjuntos iguales de imágenes mediante el protocolo *HTTP/1.1* y las otras sobre *HTTP/2*. Las imágenes fueron previamente seleccionadas, de sitios como Flickr donde sus propietarios dan autorización para el uso de estas, también se buscaron características particulares como el tamaño del archivo, ya que el objetivo es formar un conjunto donde las imágenes presenten variaciones en esta característica y de esta forma poder medir el desempeño del protocolo en diferentes escenarios.

IV. DESARROLLO

Diseño e implementación de la aplicación GTM² (Gift, Trade, Meet and More)

Esta es una aplicación web que fue desarrollada para evaluar la carga de diferentes tipos de imágenes por medio de los protocolos HTTP/1.1 y HTTP/2 para este estudio. La aplicación se ha desarrollado bajo el stack *MEAN* (*MongoDB - Express - Angular - NodeJS*) , por ser una aplicación ligera. Por otro lado, se implementaron dos servidores en NodeJS los cuales permitieron atender las peticiones que son enviadas desde el cliente, donde uno fue mediante *HTTP/1.1* y otro por *HTTP/2*, de manera simultánea. Sin embargo, para configurar el servidor con *HTTP/2* se debe realizar una serie de pasos como la creación y el uso de un certificado *SSL*, pasos que no se realizan para *HTTP/1.1* (ya que este protocolo no exige el uso de conexiones seguras).

Un certificado *SSL (Secure Sockets Layer)* es un título digital que autentifica la identidad de un sitio web y cifra con tecnología *SSL* la información que se envía al servidor [8]. El protocolo *HTTP/2* va de la mano con dicho certificado y lo usa básicamente para dar seguridad en el transporte de los paquetes. Para llevar a cabo la implementación del servidor en *HTTP/2* se debe realizar por obligación la creación o compra a priori de un certificado *SSL*. En este caso se optó por crear el certificado localmente con *OpenSSL* [9] y realizar todo el proceso para usarlo en el servidor NodeJS de la aplicación y así proceder a activar el protocolo *HTTP/2*.

Para crear el certificado SSL localmente se creó la guía "Creación de certificado SSL con OpenSSL Debian 9" [10] que muestra detalladamente los pasos que se deben seguir para crear el certificado *SSL* en cualquier distribución basada en el sistema operativo GNU/Linux Debian y así obtener la llave, el certificado y el localhost con el *HTTPS* activado.

Teniendo entonces el certificado de manera local, se procede a una parte importante de la implementación del proyecto, que es la integración del protocolo *HTTP/2* en la plataforma NodeJS, que se realizó con el uso del módulo *SPDY* [11] en el servidor. El uso de este módulo se dio debido a que *Express* como framework de NodeJS posee gran cantidad de *middlewares* que evitan el uso experimental del módulo *http2*. La implementación se evidencia en la Figura 3.

```
server.js
const APP = require('./app')
const CONFIG = require('./config')
const FS = require('fs')
const PATH = require('path')
const spdy = require('spdy')

const options = {
  key: FS.readFileSync(__dirname + '/localhost.key'),
  cert:  FS.readFileSync(__dirname + '/localhost.crt'),
  passphrase: CONFIG.passphrase
}
spdy
  .createServer(options, APP)
  .listen(CONFIG.port, (error) => {
    if (error) {
      console.error(error)
      return process.exit(1)
    } else {
      console.log(`API Rest corriendo en https://localhost:${CONFIG.port}`);
    }
})
```

Figura 3. Código fuente correspondiente a la implementación del servidor NodeJS con HTTP/2.

Donde los archivos *localhost.key, localhost.crt* y *CONFIG.passphrase*, hacen referencia a la llave privada, al certificado y a la contraseña (que se obtiene en la creación del certificado) respectivamente. Se puede también notar que el módulo *SPDY*, otorga el método *createServe*r para crear el servidor y que obtiene por parámetros los archivos antes mencionados, además del método *listen*, donde se le envía por parámetro el puerto que escuchará el servidor. De esta forma, se logra implementar el protocolo *HTTP/2* en NodeJS. La Figura 4 muestra que el protocolo *HTTP/2* ha sido configurado con éxito y que los recursos de la página se cargan mediante este.

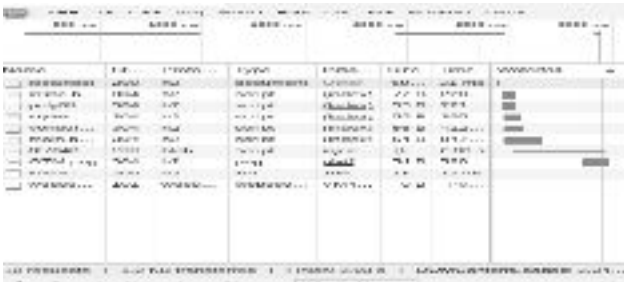

Figura 4. Conexión del protocolo *HTTP/2* establecida en el index de *GTM² corriendo en Chrome como navegador.*

Escenario de pruebas

Para efectuar con éxito las pruebas sobre el protocolo *HTTP/2* se debe establecer un escenario adecuado con el objetivo de obtener buen análisis de los resultados. Por tanto, en este caso se plantearon varios escenarios que dieron lugar a resultados que se mostrarán de manera más clara en la siguiente sección. Cabe aclarar que los computadores de los montajes estaban conectados a la misma red *LAN*. Los montajes correspondientes a los escenarios son los siguientes:

- **Escenario 1.** El primer escenario se compone básicamente de un computador portátil siendo servidor y cliente al tiempo, las características de dicho computador son: core i7, 6 GB RAM, 256 TB SSD, Debian 9 SO.
- **Escenario 2.** El segundo escenario corresponde a dos computadores de escritorio, que al igual que el escenario anterior, uno hará las veces de servidor y el otro de cliente, ambos tienen las mismas características: core i7, 8GB RAM, 1TB, Ubuntu SO.

La forma más rápida de probar la eficiencia de carga con el protocolo *HTTP/2*, es a través de contenido multimedia, y en lo posible empleando varios archivos de diferente tamaño y forma. Cabe aclarar que al establecer a una imagen como la más pequeña o la más grande es respecto a un conjunto de 50 imágenes de prueba [12]. A continuación, se describen las diferentes pruebas aplicadas:

- **Prueba 1.** Desplegar una imagen de tamaño relativamente pequeño (1.4 MB) en el servidor de Node implementando el protocolo *HTTP/1.1* (por defecto) y *HTTP/2*.
- **Prueba 2.** Desplegar una imagen de tamaño relativamente mediano (19.7 MB) haciendo las peticiones a ambos servidores.
- **Prueba 3.** Desplegar una imagen de tamaño relativamente grande (129.5 MB) haciendo las peticiones a ambos servidores.
- **Prueba 4.** Desplegar 50 imágenes de diferentes tipos de tamaño (desde 6.8 KB hasta 129.5 MB), haciendo dichas peticiones de despliegue a los servidores de Node con protocolo *HTTP/1.1* y *HTTP/2*.

Se debe tener en cuenta que cada prueba consta de 10 iteraciones y antes de ser realizadas fue borrada la caché del navegador para tener una mayor precisión en los resultados obtenidos.

Ahora, el motivo de las pruebas es principalmente ver cómo las características implementadas por el protocolo *HTTP/2*, afectan los tiempos de carga de imágenes en el navegador y presentan una solución a los problemas de *HTTP/1.1*. A continuación se procede a describir las características más importantes:

- Compresión: Como la semántica de *HTTP/1.1* se mantiene para *HTTP/2,* la sintaxis de las cabeceras sigue siendo la misma, sin embargo, los parámetros de éstas a menudo guardan los mismos valores, lo que resulta redundante. Por ello, el protocolo *HTTP/2* implementa el algoritmo llamado *HPACK,* usado para la compresión de cabeceras [2].
- Multiplexación: Para resolver el *head of line blocking*, *HTTP/2* lleva a cabo múltiples solicitudes en un único canal *TCP*. En la conexión, *HTTP/2* genera flujos independientes, donde cada flujo es único y realiza múltiples solicitudes *TCP*. De esta forma, no se requiere un orden entre las transmisiones, sin embargo, los flujos multiplexados pueden causar *HOLB*. Para prevenir esto, el cliente asigna prioridad a cada flujo. Cuando la capacidad de flujos es limitada, los flujos con mayor prioridad son enviados primero [13].
- *Server push*: Esta característica ayuda a mejorar el tiempo de carga de las páginas, se encarga de omitir solicitudes adicionales, trayendo del servidor elementos embebidos que se usarán después, sin tener que hacer más solicitudes explícitas. Es importante saber que el servidor debe implementar explícitamente esta característica [14].
- *ALPN*: es una extensión *TLS*, que puede negociar qué protocolo debería ser manejado bajo una conexión segura de modo que sea más eficiente. Es usado en el protocolo *HTTP/2* para disminuir aún más los tiempos de carga del sitio y encriptar las conexiones más rápido.
- Prioridad: Este mecanismo es usado en conexiones *HTTP/2* para permitirle al servidor priorizar archivos de *CSS* y *JS* sobre imágenes con lo que se obtiene menor tiempo de carga de página.

Cabe mencionar que las pruebas que se realizaron no cubren todas las características que brinda el protocolo *HTTP/2*, en este caso se evalúa el tiempo de carga y la multiplexación.

V. ANÁLISIS DE RESULTADOS

En esta sección se muestran los resultados de las pruebas descritas anteriormente, cómo se llevaron a cabo y qué resultados arrojaron. Mediante la presentación de los resultados se hará un análisis a las características del protocolo *HTTP/2* evaluadas.

Prueba 1

Para esta prueba se usó el escenario 1, donde en el mismo computador se corre el lado del cliente y de los servidores, dado que la imagen es de tamaño pequeño (1.4 MB) y se pudieron ejecutar las iteraciones sin mayores problemas. Según la Figura 5, se puede observar que ambas curvas correspondientes a los protocolos casi se traslapan, sin embargo, el protocolo *HTTP/2* presenta una mejoría frente al *HTTP/1.1* pero no es pronunciada. Se puede observar entonces que los tiempos de carga para una imagen de tamaño pequeño por parte de ambos protocolos es parecida y que el protocolo *HTTP/2* no muestra una gran ventaja en este escenario.

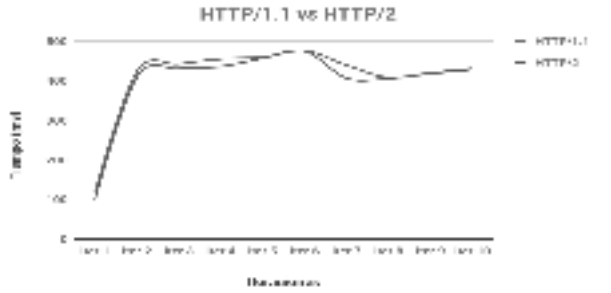

Figura 5. Tiempos de carga para una imagen de tamaño 1.4 MB en los protocolos *HTTP/1.1* y *HTTP/2*.

Prueba 2

Al igual que la prueba 1, esta prueba se realizó en el escenario 1 con la variante de que el tamaño de la imagen ya es considerado mediano (19.7 MB). La Figura 6 muestra los resultados de las 10 iteraciones hechas para dicha prueba, en la cual se puede observar que el protocolo *HTTP/2* tiene una mejoría sustancial respecto a la prueba 1 y su curva tiende a ser constante, mientras que en el protocolo *HTTP/1.1* sucede lo contrario. Se puede analizar también que en el 74% de las iteraciones el protocolo *HTTP/2* se presenta como el mejor de los dos.

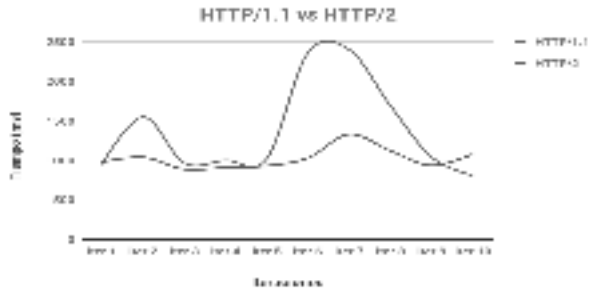

Figura 6. Tiempos de carga para una imagen de tamaño 19.7 MB en los protocolos *HTTP/1.1* y *HTTP/2*.

Prueba 3

Para una imagen de 129.5 MB, donde se utilizó el escenario 1 y se realizaron las correspondientes 10 iteraciones para dicha prueba. Los resultados se muestran en la Figura 7, donde las curvas para ambos protocolos son parecidas pero el protocolo *HTTP/2* arroja mejores tiempos de carga de la imagen. Además de que en casi todas las iteraciones el protocolo *HTTP/1.1* lanza tiempos de carga mayores a los de *HTTP/2*.

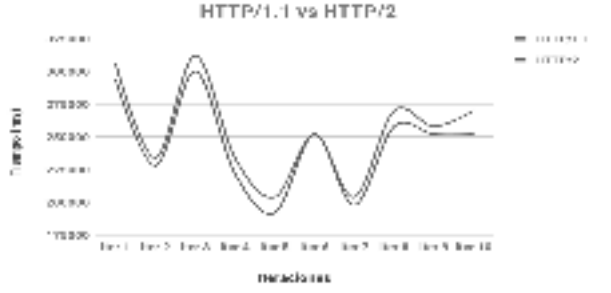

Figura 7. Tiempos de carga para una imagen de tamaño 129.5 MB en los protocolos *HTTP/1.1* y *HTTP/2*.

Prueba 4

Esta prueba se realizó en el escenario 2, ya que el flujo de datos es más grande. A diferencia de las demás pruebas realizadas, esta incluyó la carga de todas las imágenes del conjunto que se eligió inicialmente. Debido al alto flujo de peticiones las pruebas no se pudieron realizar en paralelo, así que inicialmente se hicieron las 10 iteraciones para HTTP/1.1 y luego para HTTP/2. En la Figura 8 y Figura 8.1 se muestra cómo se cargaron las imágenes en ambos protocolos. Se puede observar con detalle la multiplexación del canal *TCP* del protocolo *HTTP/2,* donde todas las imágenes cargan al tiempo desde un mismo comienzo, mientras que en *HTTP/1.1* se observa el problema de *HOLB* ya que las imágenes tienen un largo tiempo de espera en cola *(queueing)* y por ello la carga de las imágenes se da en cascada. Sin embargo, según la Figura 9, y contra toda expectativa, los tiempos de carga de *HTTP/1.1* fueron mejores que *HTTP/2* en esta prueba, esto debido a que en la carga de las imágenes por medio del protocolo *HTTP/2* se presenta un gran *TTFB* , que es el tiempo que tarda un navegador en recibir la primer pieza de información desde el servidor luego de haber hecho una petición (Figura 7.1, barra verde).

HTTP/1.1

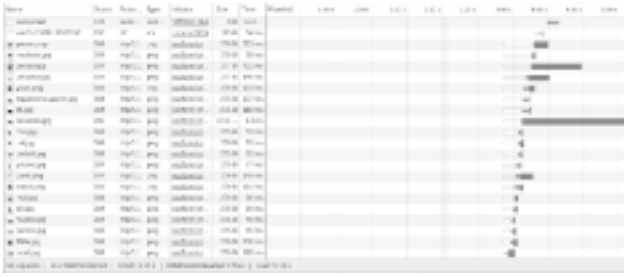

Figura 8. Tiempos de carga para todo el conjunto de imágenes en *HTTP/1.1*

HTTP/2

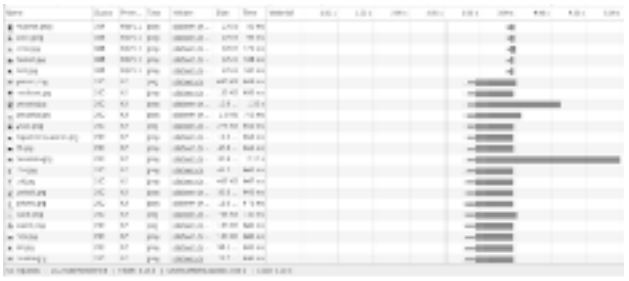

Figura 8.1. Tiempos de carga para todo el conjunto de imágenes en *HTTP/2*

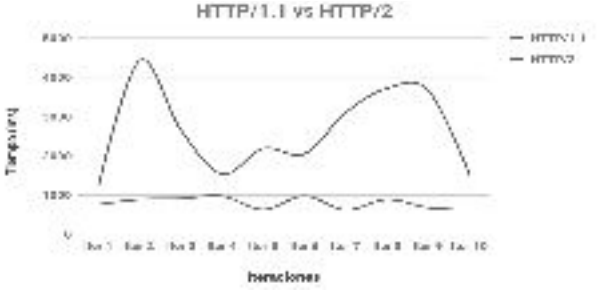

Figura 9. Tiempos de carga para un conjunto de imágenes de tamaño desde 6.8 KB hasta 129.5 MB en los protocolos *HTTP/1.1* y *HTTP/2*.

Como hipótesis se tiene que estos resultados se deben principalmente a dos factores, el primero es que la

configuración de *HTTP/2* prioriza las peticiones de manera que las peticiones del contenido multimedia se atienden casi al final por ello se ve un aumento en el *TTFB*. Por otro lado, para implementar el protocolo en el servidor, se está usando un módulo de terceros en NodeJS que posiblemente tiene problemas de implementación. Por lo anterior, se decide proponer otro tipo de pruebas por subconjuntos donde las imágenes se agrupen de acuerdo a un intervalo y que sus límites están dados por el tamaño que es la característica que más interesa hasta el momento. Entonces, se decide tomar 33 de las 50 imágenes (las más pesadas) y después dividir dicha cantidad en 3 subconjuntos (cada uno de 11), donde se le realiza a cada uno, pruebas similares a las anteriores.

- **Prueba 5.1.** Desplegar un conjunto de 11 imágenes que tienen un tamaño que va desde 1.4 MB hasta 8.3 MB.
- **Prueba 5.2.** Desplegar un conjunto de 11 imágenes que tienen un tamaño que va desde 10.8 MB hasta 19.7 MB.
- **Prueba 5.3.** Desplegar un conjunto de 11 imágenes que tienen un tamaño que va desde 23.1 MB hasta 129.5 MB.

Prueba 5.1, 5.2 y 5.3

Los resultados obtenidos en las pruebas 5.1 y 5.2 son muy satisfactorios ya que como se observa en las Figuras 10 y 11 el protocolo *HTTP/2* es 75% y 84% respectivamente más rápido que *HTTP/1.1*, y aunque en algunos puntos son casi iguales o se interceptan el protocolo que ha sido objeto de estudio sigue arrojando, en promedio, muy buenos resultados. Caso contrario sucede en la prueba 5.3, que le da más fuerza a la hipótesis de que las peticiones tardan más cuando el conjunto de objetos es muy pesado. En la Figura 12 se puede observar que el comportamiento es casi igual al de la prueba 4 (Figura 9) donde los tiempos de carga de *HTTP/1.1* son mínimos respecto a *HTTP/2*.

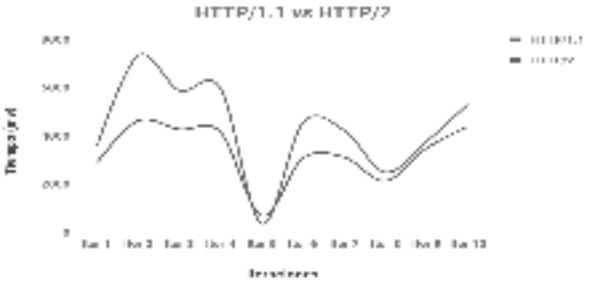

Figura 10. Tiempos de carga de un subconjunto de imágenes pequeñas en *HTTP/1.1* y *HTTP/2*.

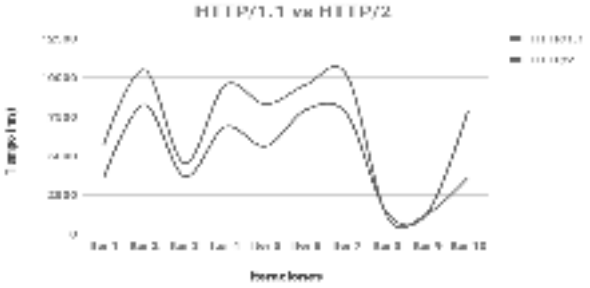

Figura 11. Tiempos de carga de un subconjunto de imágenes intermedio en *HTTP/1.1* y *HTTP/2*.

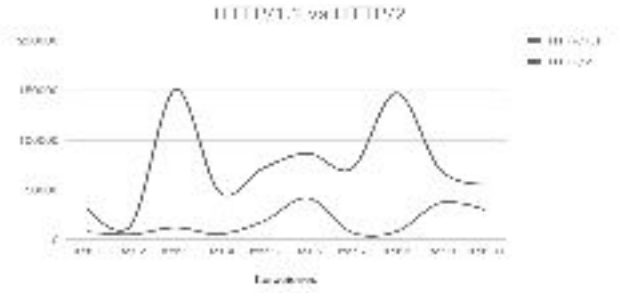

Figura 12. Tiempos de carga de un subconjunto de imágenes grande en *HTTP/1.1* y *HTTP/2*.

Finalmente, los promedios obtenidos de las pruebas se plasman en la Tabla 1, en la cual se pueden observar, a modo de resumen, los resultados cuantitativos que van ligados con las gráficas expuestas anteriormente y que en el comparativo general entre los protocolos, *HTTP/2* se comporta mejor que su antecesor. Sin embargo, también se observa los grandes tiempos de carga que arroja *HTTP/2* cuando se evalúan grandes conjuntos de imágenes, siendo de esta forma superado por el protocolo *HTTP/1.1*.

Tabla 1. Tiempos promedio en pruebas de *HTTP/1.1* y *HTTP/2*.

Prueba	Tiempo promedio (ms)		Diferencia entre tiempos (ms)
	HTTP/1.1	HTTP/2	
1	407	397	10
2	1378	1024	354
3	254400	244800	9600
4	616	2608	-1990
5.1	4334,8	3275	1059,8
5.2	6430	5429	1001
5.3	17470	74797	-57327

VI. CONCLUSIONES Y TRABAJO FUTURO

Es necesario aclarar que este trabajo se realizó con el ánimo de analizar algunas de las características del protocolo *HTTP/2* como la multiplexación y los tiempos de carga sobre varios conjuntos de imágenes en formato *JPEG* de diferente tamaño que hacen de dicho protocolo un estándar prometedor. Por tanto, el principal objetivo propuesto fue plantear un escenario de pruebas donde se puedan evaluar algunas de estas características y realizar las pruebas necesarias sobre las imágenes para observar su comportamiento, comparándolo con el protocolo *HTTP/1.1* usando un servidor NodeJS. Se puede concluir que los resultados de las pruebas varían, y de modo general el protocolo *HTTP/2* supera a *HTTP/1.1* entregando mejores tiempos de carga en promedio, sin embargo, en pruebas que requieren una carga de gran cantidad de imágenes, *HTTP/1.1* se comporta mejor que el nuevo protocolo. De acuerdo a esto, no sólo el tamaño de la imagen que se carga es un factor influenciador, sino que también la cantidad de imágenes que se cargan incide en el promedio del tiempo final. Agregado a esto, se debe tener en cuenta la integración del módulo *SPDY* al servidor NodeJS, ya que si bien se incorporó al

proyecto de manera exitosa y permitió la activación del protocolo *HTTP/2* en la aplicación, no tuvo el comportamiento esperado como lo muestra el resultado de las pruebas, por ello se plantea como trabajo futuro investigar más a fondo sobre otros frameworks que trabajen sobre *NodeJS* y tengan un módulo de *HTTP/2* implementado para analizar y comparar resultados, y así determinar cuáles son los mejores módulos de este protocolo para trabajar en *NodeJS*. También se plantea buscar la forma de evaluar las nuevas características que el protocolo trae. Finalmente, con la ayuda de las pruebas realizadas se pudo observar que el uso del protocolo no asegura más velocidad en los tiempos de carga, en ocasiones algunas páginas que usan HTTP/1.1 pueden tener tiempos iguales o mejores ya que se encargan de configurar adecuada y rigurosamente sus servicios por medio de trucos que algunos entusiastas de la programación han hecho durante este periodo de estancamiento para intentar corregir algunas fallas, por ello se recomienda hacer un adecuado y profundo análisis sobre los costos y beneficios que traería migrar una plataforma a HTTP/2.

VII. REFERENCIAS

[1] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999). RFC 2616-Hypertext Transfer Protocol-HTTP/1.1. Disponible en: https://tools.ietf.org/html/rfc2616

[2] de Saxcé, H., Oprescu, I., & Chen, Y. (2015, April). Is HTTP/2 really faster than HTTP/1.1?. *2015 IEEE Conference on*(pp. 293-299). Disponible en: https://ieeexplore.ieee.org/document/7179400/

[3] Belshe, M., & Peon, R. (2015, May). Hypertext Transfer Protocol Version 2 (HTTP/2) (M. Thomson, Ed. IETF). Disponible en: https://tools.ietf.org/html/rfc7540

[4] Friedl, S., Popov, A., Langley, A., & Stephan, E. (2014). Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. Disponible en: https://www.rfc-editor.org/rfc/pdfrfc/rfc7301.txt.pdf

[5] Belshe, M. and Peon, R. (2012). draft-mbelshe-httpbis-spdy-00 - SPDY Protocol. [online] Tools.ietf.org. Dispobible en: https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[6] Zarifis, K., Holland, M., Jain, M., Katz-Bassett, E. and Govindan, R. (2016). [online] semanticscholar.org. Disponible en: https://pdfs.semanticscholar.org/c274/8de616247677b44815c6a7477afed5c79084.pdf

[7] Schwaber, K., & Sutherland, J. (2013). La guía de Scrum. La guía definitiva de scrum: Las reglas del juego.

[8] Freier, A., Karlton, P. and Kocher, P. (2011). RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0. [online] Tools.ietf.org. Disponible en: https://tools.ietf.org/html/rfc6101

[9] Young, E. A., Hudson, T. J., & Engelschall, R. S. (2001). OpenSSL. World Wide Web, http://www. openssl. org/

[10] Cadavid, S. & Giraldo, J. (2018). Creación de certificado SSL con OpenSSL Debian 9. [online] Google Docs. Disponible en: https://docs.google.com/document/d/1frJsDOHCTd-lOoDRhPUFxJGDwR80Wbq4VhzUuBcB3Go/edit?usp=sharing

[11] Dias, D., Indutny, F. and Rauch, G. (2015). spdy. [online] npm. Disponible en: https://www.npmjs.com/package/spdy

[12] users, Flickr. (2005). Very large photos. [online] Flickr. Dispobible en: https://www.flickr.com/groups/veryverylargephotos/pool/with/33522578970

[13] Kim, H., Lee, J., Park, I., Kim, H., Yi, D. H., & Hur, T. (2015, August). The upcoming new standard HTTP/2 and its impact on multi-domain websites. In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific* (pp. 530-533). IEEE.

[14] Zimmermann, T., Rüth, J., Wolters, B., & Hohlfeld, O. (2017, June). How HTTP/2 Pushes the Web: An Empirical Study of HTTP/2 Server Push. In *Semantic Scholar*.