Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de las Telecomunicaciones

Optimización de Páginas Web: Visión teórica y análisis práctico

Autor: Adrián Vázquez Sanisidro
Tutor: Francisco José Fernández Jiménez

Departamento Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de las Telecomunicaciones

Optimización de Páginas Web: Visión teórica y análisis práctico

Autor:
Adrián Vázquez Sanisidro

Tutor:
Francisco José Fernández Jiménez
Profesor colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Optimización de Páginas Web: Visión teórica y análisis práctico

Autor: Adrián Vázquez Sanisidro

Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

Este proyecto nunca habría sido posible sin la ayuda de mi tutor Francisco José Fernández Jiménez. También quiero agradecer a Ilya Grigorik por sus enriquecedoras charlas, su libro "High Performance Browser Networking" y por molestarse en responderme las dudas que le hice llegar.

Por otro lado, agradecer a mis padres por su educación y apoyo sin los cuales este trabajo no hubiese sido posible. También a María Laó por soportarme, acompañarme en mis viajes y ayudarme a terminar este grado en ingeniería.

No puedo dejar sin nombrar a Dunai Fuentes, a Alberto Almenara y a Guillermo Roche que han sido quienes me han aguantado mientras este proyecto pasaba de ser una idea a lo que es ahora.

Y por último agradecer a todos mis antiguos profesores por su enseñanza a lo largo de los años, además de a todos mis amigos por enriquecerme como persona.

Adrián Vázquez Sanisidro

Sevilla, 2017

Resumen

Este trabajo pretende servir como base a aquellas personas que quieran introducirse en el mundo de la optimización de páginas webs, centrándose en las técnicas más básicas pero basándose a su vez en las últimas tecnologías disponibles y mirando hacia el futuro con las novedades que están a punto de llegar.

El trabajo se divide principalmente en 3 partes:

- La primera parte es una aproximación más teórica a la temática, donde empezamos explicando el estado del arte centrándonos en cómo medir y en las herramientas más utilizadas durante la última década para valorar el nivel de optimización de una página web. A continuación, explicamos cómo configurar la capa de aplicación correctamente con HTTP/2, y cómo optimizar los recursos adicionales de una página web, tales cómo CSS, JavaScript, fuentes e imágenes, además de la compresión de todos ellos.
- En la segunda parte, medimos los cambios resultantes tras hacer uso de varias de las técnicas comentadas, en la parte anterior, sobre una página web real. Por un lado, las aplicamos de forma individual y posteriormente realizamos simultáneamente las modificaciones más útiles para conseguir la mejor optimización posible. Combinando todas estas técnicas hemos conseguido reducir el tamaño de la página web en más de un 70%, haciendo que el tiempo de carga completa se haya reducido a la mitad. Todo esto sin alterar el diseño, ni ninguna función de la página web.
- Para finalizar tenemos los anexos, donde se explica como instalar una utilidad de medida y se encuentra el código necesario que hemos desarrollado en NodeJS para ponerla en funcionamiento de forma automatizada a través de su API. Además, ponemos un ejemplo de como utilizar una nueva tecnología de compresión muy útil, llamada BROTLI, que aún no se está utilizando de forma extensa debido a la dificultad que presenta a la hora de integrarse en los servidores que se utilizan en la actualidad.

Abstract

This project aims to be a base for those people who want to enter the world of Web Page Optimization (WPO), focusing on the most basic techniques but based on the latest technologies available and looking to the future with the latest innovations that are about to arrive.

It is divided mainly in three blocks:

• The first block is a theoretical approach to WPO, where we start explaining the state of the art, focusing on how to measure and the most used tools during the last decade to assess the level of optimization of a web page. Next, we explain how to properly configure the application layer with HTTP/2, and how to optimize the additional resources of a web page, such as CSS, JavaScript, fonts and images, in addition to compressing all of them.

• In the second part, we measure and apply several of the techniques discussed in the previous section on a real web page. We apply these changes individually and after that, we make them simultaneously to achieve the best possible optimization. Combining all these techniques we have managed to reduce the size of the website by more than 70% and the full load time has been cut in half.

• Finally, we have the appendices, which explains how to install a measurement utility. There is also, the code that we have developed in NodeJS to make all the measurements in an automated way through its API. In addition, we give an example of how to use a very useful new compression technology, called BROTLI, which is not yet being used extensively due to the difficulty that presents at the time of the integration in the servers that are currently used.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

The web should be fast.

- Matt Cutts -

Actualmente en un mundo tan competitivo como el de las páginas webs, es muy importante ofrecer al usuario el contenido que busca, pero para poder retenerlo y rentabilizarlo es aún más importante hacerlo rápido. Especialmente en un ambiente lleno de dispositivos móviles, donde tus visitantes no quieren perder tiempo esperando que una página web cargue.

Cuando un visitante entra en una página web, el tiempo percibido de carga es el llamado *end user response time.* La mayor parte de este tiempo, a diferencia de lo que se suele pensar, no ocurre en el servidor (*backend*) cuando está generando la página, sino que ocurre en el navegador del usuario (*frontend*) mientras descarga el HTML y realiza las peticiones y procesa las hojas de estilo CSS, los archivos Javascript, las fuentes y las imágenes.

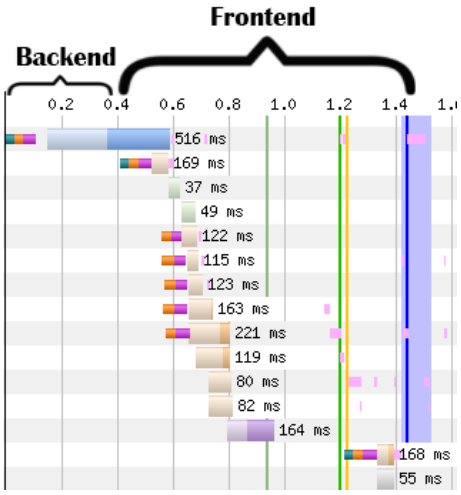


Figura 1-1. Gráfico de carga de linkedin.com mostrando que parte es frontend y backend

Por ello nos centraremos principalmente en la optimización de la conexión y del *frontend* que es más universal y tiene un mayor potencial, debido a que el backend depende mucho de la aplicación que estemos utilizando en el servidor y sería complejo llegar a conclusiones útiles de forma generalista.

1.1 Estructura del trabajo

Empezaremos viendo el estado del arte: cómo y qué debemos medir, las herramientas más importantes que nos pueden ayudar y las últimas mejoras que han ocurrido en la última década en la capa de aplicación HTTP.

Después nos centraremos en la mejor forma de optimizar su funcionamiento y que cosas debemos evitar. Tras ello pasaremos a un caso práctico en el que optimizaremos una página web al completo.

Las acciones que se suelen realizar para optimizar una página web son las siguientes:

- En el servidor:
    - Uso de cache en el servidor si se está usando una aplicación dinámica, es muy sencillo de hacer e importante de utilizar en las aplicaciones web más usadas actualmente (Wordpress, Prestashop, Magento, etc)
    - Soporte de nuevos protocolos (SPDY o HTTP/2)
    - Enviar cabeceras para que los navegadores mantengan el contenido en cache
    - Evitar redirecciones
- En el código HTML de la web:
    - Compresión HTML
    - Ordenar correctamente la carga de los archivos externos para optimizar la carga y evitar redibujados de la página web, especialmente evitar incluir archivos JavaScript externos antes de la carga de la parte visible de la web, ya que estos archivos bloquean el renderizado de la misma mientras se descargan y procesan.
- En los archivos adicionales:
    - Minificación y unión de multiples archivos CSS y JavaScript, útil si utilizamos HTTP/1.x.
    - Usar imágenes del tamaño con el que se van a mostrar.
    - Utilizar el formato de imagen adecuado y optimizarlas
    - Comprimir los archivos CSS y JavaScript

Tras analizar todos los puntos indicados y otros adicionales, pasaremos a realizar mediciones a una página web real basada en Wordpres y aplicaremos las optimizaciones necesarias de forma independiente para ver el efecto que tiene cada una de ellas sobre su rendimiento. Posteriormente optimizaremos la misma página web por completo para obtener el máximo rendimiento posible.

En los anexos podemos encontrar como instalar y configurar una instancia privada de WebPageTest, que es una herramienta de medición muy útil como se explicará en el capítulo 3, también se explica la instalación de las dependencias necesarias y el código del script utilizado para automatizar la recolección de los datos de las mediciones, y por último se explica cómo instalar la herramienta adecuada para poder usar el algoritmo de compresión BROTLI, el cúal está explicado en el punto 4.3.2.

2 MOTIVACIÓN Y OBJETIVOS

Mobile is important, and coming faster than most people in this room realize.

- Matt Cutts, 2014 -

El mundo cambio el día que Matt Cutts, el exdirector del departamento de Calidad de búsquedas de Google explico que la velocidad de carga de las páginas webs iba a ser incorporado en su algoritmo de posicionamiento. En ese instante empezó la carrera para tener la página web más optimizada posible y dejo de ser algo secundario en las agendas de las empresas. Empezarón a surgir nuevas herramientas para ayudar a conseguir hacer las páginas webs más rápidas y aparecieron nuevos empleos centrados en la optimización de páginas webs (Conocida habitualmente como WPO por sus siglas en inglés).

2.1 Motivación

Las empresas muestran cada vez más atención a la velocidad de carga de sus páginas web y a su optimización, debido a que tan solo 1 segundo extra pueden ser miles de clientes perdidos.

Esta más que demostrado que las páginas webs más rápidas consiguen una mejor retención de los usuarios, usuarios más contentos y mayores conversiones. Diversas empresas han dado información sobre cuanto pierden cuando su web se vuelve más lenta:

- Google: 400ms de retraso en la carga lleva a una caída del 0,59% en las búsquedas por usuario [1]
- Shopzilla: Disminuyendo el tiempo de carga en 5 segundos, consiguieron aumentar entre un 7 y un 12% la tasa de conversión, duplicar el tráfico proveniente de buscadores y ahorrar un 50% en servidores [2]
- Netflix: Usando una sola optimización, activar la compresión GZIP, consiguieron aumentar más de un 17% la velocidad de su web y redujeron su tráfico saliente un 50% [3]

En definitiva, que la velocidad de carga de una página web sea buena no es algo opcional, sino algo necesario, y por eso se está creando toda una nueva industria alrededor de su medición y mejora.
En los últimos años han surgido varias propuestas de nuevas implementaciones para optimizar y mejorar el funcionamiento del antiguo protocolo HTTP/1.1, de las cuales hablaremos en el próximo capítulo.

Además, actualmente, Google tiene muy en cuenta la velocidad de carga de las páginas web, y es un factor cada vez más importante dentro de su algoritmo de posicionamiento. [4]

Es complicado poder encontrar un estudio serio y completo sobre el tema, ya que solo hay comparaciones parciales y pequeñas guías con generalidades sobre cómo conseguir mejorar una página web.

2.2 Objetivos

La finalidad de este trabajo es principalmente diagnosticar de forma general cuales son, en la actualidad, las mejores practicas para conseguir hacer que una página web sea más rápida, centrandonos en aspectos que se suelen descuidar cómo, por ejemplo, la mejor forma de configurar la capa de aplicación para inicializar la conexión lo más rápido posible.

El objetivo final de este trabajo es que el lector acabe teniendo suficientes nociones básicas de optimización de páginas webs como para poder mejorar por si mismo la velocidad de carga de sus páginas web.

3 ESTADO DEL ARTE

All communication should be secure, always, and by default! HTTPS everywhere!

- Ilya Grigorik -

Si queremos que algo funcione más rápido, el primer paso en el que debemos centrarnos es en ser capaces de medir cómo de rápido es actualmente, además de reflexionar cómo y qué queremos medir. Un posible problema de no llevar a cabo la fase de reflexión es acabar midiendo para obtener el resultado que buscábamos, y esto es algo que se debe intentar evitar por todos los medios ya que no obtendríamos buenas conclusiones. Por lo que es importante estandarizar los tiempos que vamos a tener en cuenta.

3.1 Tiempos que debemos medir

Respecto a los tiempos que son importante medir y tener en cuenta en la carga de una página web, se pueden diferenciar el tiempo del inicio de la conexión, por un lado, y los tiempos de percepción de carga, que incluyen a los primeros, por otro.

El tiempo de inicio de la conexión lo podemos subdividir en:

- El tiempo de la redirección, en caso de que exista.
- Tiempo de conexión, incluye el tiempo de la resolución de DNS y la conexión TCP, además de la negociación TLS, si es una web segura
- Tiempo en el Backend, es el tiempo que tarda el servidor en procesar nuestra petición desde el momento que la conexión se ha completado, hasta que empieza a respondernos.

Los tiempos de percepción de carga más importantes son:

- TTFB o Tiempo hasta el primer Byte, es el tiempo que transcurre desde que se envía la petición hasta que el navegador recibe el primer Byte del html.
- Start Render o Tiempo del primer renderizado, es el tiempo que transcurre hasta que el primer elemento no blanco es mostrado en la pantalla del navegador.
- Load Time o Tiempo de carga, es cuando se produce el evento Onload de JavaScript, en este punto todo el contenido de la página debería estar cargado incluyendo las imágenes y sólo faltarían los eventos de JavaScript que se ejecutan en este punto.
- Fully Loaded o Tiempo de carga completa, es el tiempo que transcurre hasta que la página está completamente cargada

También es muy importante tener en cuenta, aunque no es un tiempo propiamente dicho, el *Speed Index* [5] o Índice de carga, que es un algoritmo que nos indica la percepción que tendrá un usuario del tiempo de carga de nuestra página web y que tiene en cuenta los cambios visuales durante la carga y lo completa que se muestra la web en la pantalla respecto al resultado final. En este caso, es importante señalar que cuanto menor sea, mejor percepción de velocidad ofrece nuestra página web.

A continuación, explicamos gráficamente su funcionamiento usando dos mediciones, del tiempo de carga de una página web, que tuvieron el mismo Tiempo de carga, pero un comportamiento diferente durante el mismo:

Figura 3-1. Carga de una misma página web con distintas velocidades de pintado

Si pintamos una gráfica con el porcentaje visualmente pintado en pantalla en el eje vertical y tiempo en el eje horizontal, obtenemos lo siguiente:

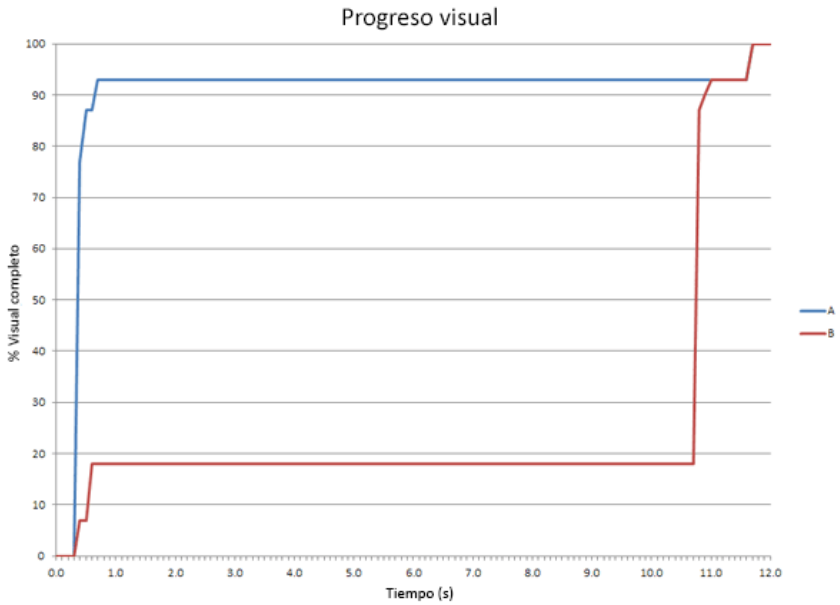


Figura 3-2. Gráfico de progreso visual de la carga

Tomando este gráfico como ejemplo, el Indice de carga o Speed Index, de cada uno de los dos ejemplos, sería la integral sobre la curva de cada uno de ellos.

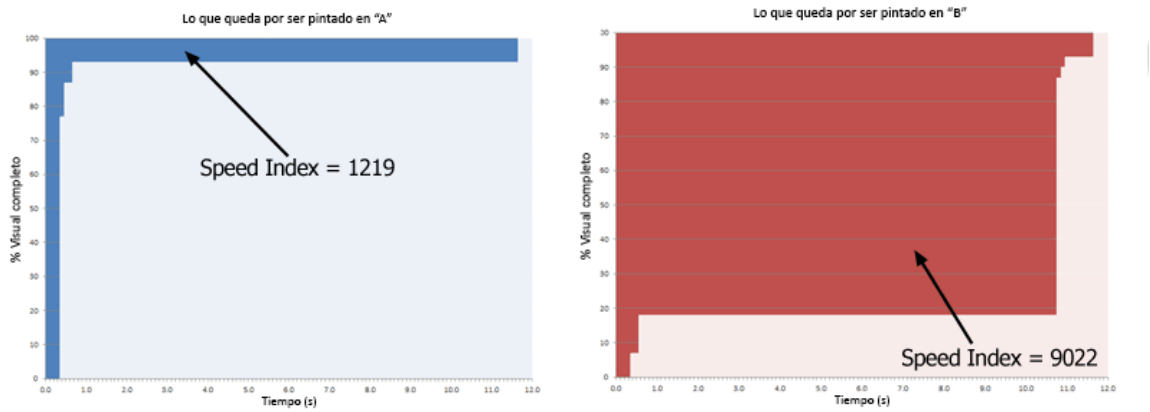


Figura 3-3. Ejemplo gráfico del SpeedIndex

La podemos calcular usando la siguiente fórmula:

$$Speed\ Index = \int_0^{TCarga} 1 - \frac{\%VC}{100} \tag{3–1}$$

Siendo %VC el porcentaje visual completo y Tcarga el tiempo de carga en milisegundos.

3.2 Herramientas de medición sintéticas

Las llamadas herramientas sintéticas nos permiten monitorizar una página web usando un entorno muy controlado y prácticamente ideal, por eso, estas medidas son sólo una aproximación al rendimiento real de la carga de una página web en el navegador de nuestros visitantes y nos pueden servir como referencia para realizar cambios y mejoras en ella.

Las herramientas sintéticas tienen la ventaja que son muy fáciles de usar, son gratis, no necesitan implementar ningún código y tampoco necesitan que la página web tenga tráfico. Son las herramientas ideales durante la creación de la web. Las más usadas son:

- WebPageTest [6], permite modelar la conexión hasta el más mínimo detalle (velocidad de carga, descarga, latencia, etc), permite obtener las trazas TCP (usando tcpdump) e incluso modificar el comportamiento del navegador. Permite realizar las medidas en Chrome, Firefox, la mayoría de las versiones de Internet Explorer y en dispositivos móviles. También tiene una API para realizar mediciones masivas de forma sencilla y además es Open Source por lo que puedes crear tu propio servidor para realizar medidas. Está es la herramienta que hemos utilizado en este trabajo para hacer las mediciones, y explicamos como montar tu propio servidor en el Anexo A.
- GTMetrix [7], tiene menos opciones, pero comprueba de forma rápida si la web cumple con las buenas prácticas indicadas en Google Pagespeed y en YSlow de Yahoo.
- Lighthouse [8], a diferencia de las demás es una extensión para Chrome aunque está disponible para usar como una herramienta de línea de comandos. Nos permite realizar múltiples tests a una web. Además de las pruebas de rendimiento donde indica datos muy interesantes como el tiempo del primer renderizado de la web, la lista de los recursos críticos que bloquean el inicio del renderizado, y las posibles mejoras que se podrían aplicar (indicando el tiempo de ahorro estimado). También hace comprobaciones de la checklist de las Aplicaciones Web Progresivas para móviles que Google presento recientemente.
- Herramientas para Desarrolladores de Chrome, nos permite ver como ocurre la carga de una web en nuestro ordenador utilizando nuestra conexión. Es muy útil para entender que está pasando y para diagnosticar problemas.

3.3 Herramientas de medición sintéticas para móviles

El último año, el tráfico web móvil sufrió un incremento del 63% [9] y se esperá que siga creciendo al mismo ritmo durante, al menos, los próximos 4 años.

Debido a esto, además de asegurarnos que nuestras webs se visualizan de forma correcta en dispositivos móviles, es importante asegurarnos que también cargan adecuadamente en los mismos. Los dispositivos móviles tienen grandes restricciones de procesamiento y en la cantidad de datos que pueden utilizar. Además, las redes móviles tienen una alta latencia, aunque con las nuevas redes 4G se están consiguiendo grandes mejoras en este punto.

Una página web lenta o que cargue más recursos de los necesarios, proporciona una experiencia de usuario especialmente pobre en este tipo de dispositivos, y por ello es importante asegurarse que esto no ocurre.

Para realizar mediciones móviles existen dos tipos de herramientas:

3.3.1 Emulación de escritorio

La mayoría de las webs detectan que el visitante está usando un navegador móvil por el tamaño de la pantalla,

usando CSS media queries [10]. Para engañar al navegador sólo hay que cambiar el tamaño del viewport [11] para que sea tan pequeño como el de un dispositivo móvil.

El resto de las webs utilizan el User-Agent, incluido en la petición, el cual puede ser fácilmente modificado por el de un móvil. Este método puede ser utilizado para la fase de diseño, pero no es aconsejable utilizarlo para medir la velocidad en dispositivos móviles debido a la baja capacidad de procesamiento de los mismos comparados con un ordenador.

3.3.2 Dispositivos móviles nativos

En este caso se utilizan dispositivos reales para hacer las pruebas, es común la utilización de móviles Android para las pruebas ya que gracias a la posibilidad de obtener permisos de administrador es más sencillo utilizar aplicaciones que permitan medir todas las partes de la petición. Debido a su alto uso también se están desarrollando herramientas que usan iphones para las mediciones. [12]

Chrome para Android ofrece la posibilidad de realizar mediciones fácilmente conectando tu móvil a un ordenador que ejecute Chrome y utilizando las Herramientas para Desarrolladores.

3.3.3 Emulación de dispositivos móviles

La emulación consiste en simular un dispositivo móvil utilizando un PC y usarlo para realizar las mediciones. No hemos encontrado ningún programa que lo soporte oficialmente, aunque algunas personas han conseguido realizar medidas con WebPageTest utilizando la emulación en dispositivos iOS y en dispositivos Android , aunque debido a la mayor dificultad que los otros dos métodos de medición sintética para móviles, actualmente no parece ser un campo con futuro.

3.4 Herramientas RUM (Monitorización de Usuarios Reales)

Las herramientas RUM nos permiten ver como se ha comportado la carga de una página web en los navegadores de los visitantes a nuestra página, con dicha información podemos sacar estadísticas para ver fácilmente qué puntos están fallando en la mayoría de ellos y ver donde se puede mejorar.

Son las herramientas perfectas para usar en webs con un tráfico elevado, donde podremos monitorizar a una muestra de nuestros visitantes y sacar conclusiones muy acertadas sobre cómo mejorar el rendimiento y proporcionar una mejor experiencia a los usuarios. Además, nos permiten monitorizar fácilmente problemas en flujos de compra y otras operaciones que nuestros usuarios puedan realizar en nuestra web.

Existen muchos tipos de herramientas RUM pero la mayoría son de pago (SaaS) y requieren modificar la web añadiendo un pequeño código javascript que envía los datos del rendimiento a un servidor externo, por lo que la implementación en algunos casos no es completamente trivial. Algunas de las más usadas son New Relic Browser [13], Soasta mPulse [14] o Pingdom Performance Monitoring [15].

3.4.1 Cómo modelar las conexiones

Para la mayor parte de las webs, es suficiente con el uso de herramientas sintéticas, ya que haciendo un buen modelado de las conexiones de nuestro público objetivo podemos obtener unos datos muy realistas.

Por ejemplo, si nos encargan optimizar la parte de la web de una universidad donde los alumnos miran su horario y el lugar donde tienen clase, podemos suponer con bastante acierto, que la mayoría usará sus móviles conectados a la red de la universidad (con una buena conexión). Un buen test sería modelar la prueba usando un navegador web móvil, desde un servidor de pruebas que este cerca de la universidad.

Por ello, el primer paso antes de medir es estudiar de donde son los usuarios que visitan nuestra web y como es su conexión, para ello podemos apoyarnos en herramientas de analítica web como Google Analytics [16] o Piwik [17].

3.5 Google PageSpeed y Yahoo Yslow

Yslow fue una de las primeras herramientas que se crearon para ayudar a optimizar la velocidad de las páginas webs, su desarrollador Steve Souders la creo en 2007 [18] para ayudarse en su trabajo como jefe de optimización en Yahoo. Su funcionamiento es muy sencillo ya que sólo comprueba si una web cumple ciertas reglas preestablecidas. Posteriormente en 2012, apareció Yslow 2.0 donde se revisó el conjunto de reglas, se mejoró la flexibilidad de Yslow y se convirtió en un proyecto multiplataforma de código libre.

En Julio de 2011, apareció impulsado por Google el proyecto PageSpeed que actualmente es el punto de referencia a la hora de optimizar una página web. Al igual que Yslow, es una herramienta que comprueba el cumplimiento de ciertas reglas, aunque se centra principalmente en las cuestiones más importantes que afectan al rendimiento en los navegadores modernos, y algunas de las reglas son más complejas de comprobar.

3.5.1 Reglas de Yslow 2.0

Estas reglas [19] fueron pensadas con el funcionamiento de HTTP/1.x en mente, como veremos más adelante algunas actualmente no tienen mucho sentido si utilizamos HTTP/2.

- Reducir la cantidad de peticiones HTTP.
- Reducir las búsquedas DNS a realizar sirviendo todo desde la mínima cantidad de dominios posible.
- Evitar las redirecciones para llegar a la página destino.
- Las llamadas Ajax repetidas deben guardarse en cache.
- Retrasar la carga de los recursos que no hagan falta inicialmente.
- Precargar recursos cuando sea posible.
- Reducir el número de elementos en el DOM.
- Separar componentes con dominios.
- Minimizar el número de Iframes, lo deseable sería no tener ninguno.
- Evitar los errores 404, ya que ofrecen una mala experiencia a los usuarios.
- Usar una CDN (Content Delivery Network) para los recursos estáticos.
- Agregar las etiquetas Expires o Cache-Control Header en la cabecera HTTP.
- Usar la compresión Gzip.
- Configurar correctamente las ETags o desactivarlas.
- Vaciar el Buffer rápido para que el html empiece a llegar al cliente lo antes posible.
- Usar GET para peticiones AJAX.
- Poner los archivos JavaScript al final del HTML.
- Usar archivos externos para el JavaScript y el CSS.
- Minificar los archivos JavaScript y CSS, quitando espacios en blanco y líneas redundantes.
- Eliminar los scripts duplicados.
- Minimizar los accesos a DOM.
- Delegación de eventos, es aconsejable agrupar los eventos JavaScript que se utilicen. Ya que una gran cantidad de los mismos ralentizan la experiencia de usuario.
- Poner las hojas de estilo CSS en la parte superior del html.
- Evitar expresiones CSS complejas.
- Evitar utilizar @import en los archivos CSS, ya que implican una petición adicional después de procesar

el archivo CSS.

- Evitar el uso del filtro *AlphaImageLoader* para cargar imágenes PNG semitransparentes, ya que este filtro bloquea el renderizado.
- Reducir el tamaño de las cookies.
- Poner si es posible las imágenes y archivos JavaScript en dominios sin cookies.
- Optimizar las imágenes.
- Juntar las imágenes pequeñas en *Sprites CSS*.
- No escalar imágenes en HTML, es decir usar imágenes en el tamaño con el que se van a mostrar.
- Utilizar un fichero favicon.ico, que sea ligero y se pueda guardar en cache.
- Evitar utilizar recursos de tamaño superior a 25kb, debido a que los primeros iPhone no guardaban en cache archivos de mayor tamaño.

3.5.2 Reglas de Google PageSpeed

Actualmente se dividen en dos grupos, las reglas de velocidad y las reglas de usabilidad. [20]

Las reglas de velocidad son:

- Evitar los redireccionamientos para llegar a la página de destino.
- Habilitar compresión *DEFLATE* o *gzip* para todos los recursos de la página. Aunque actualmente existe una forma más eficiente de compresión que trataremos en el siguiente capítulo.
- Mejorar el tiempo de respuesta del servidor hasta que sea inferior a 200ms.
- Especificar cabeceras para caché de navegador en los recursos estáticos.
- Minificar recursos HTML, CSS y JavaScript.
- Optimizar las imágenes.
- Optimizar la entrega de CSS, insertando en la cabecera el código CSS esencial y retrasando el resto como archivos externos después de la finalización del html.
- Priorizar el contenido visible, reduciendo el contenido necesario para mostrar la parte superior de la página de forma que quepa en el tamaño de la primera ventana de congestión de TCP.
- Quitar el JavaScript que bloquea la carga, es decir aplazar las llamadas a recursos JS externos de forma síncrona y si es muy pequeño insertarlo en el código html; debido a que el navegador espera recibir el archivo JS externo antes de continuar con la carga.
- Usar recursos JavaScript asíncronos siempre que sea posible.

Y las Reglas de usabilidad son:

- Evitar los plugins como Flash, Silverlight y Java, ya que muchos navegadores no los soportan y suelen ralentizar la carga.
- Configurar la ventana gráfica, ya que controla cómo se muestra una página web en un dispositivo móvil y si no está configurado se mostrará igual que se muestra en un PC.
- Adaptar el contenido a la ventana gráfica, para evitar a los usuarios tener que desplazarse horizontalmente.
- Aplicar el tamaño adecuado a los botones táctiles para poder pulsarlos con un dedo fácilmente.
- Utilizar tamaños de fuente que se puedan leer.

3.6 Capa de Aplicación: HTTP

Actualmente una gran cantidad de páginas web utilizan la anticuada tecnología HTTP/1.1 definida en la RFC 2616 que data del año 1999, y que aunque ha demostrado ser útil y sólida, no es perfecta. Por ello en la última década se ha avanzado hacia su mejora, primero mediante el protocolo SPDY que trabaja en conjunto con HTTP/1.1 pero añadiéndo nuevas opciones, y posteriormente con HTTP/2 que supone un rediseño total de la forma en la que se transmite la información, pero manteniendo los métodos, códigos de estado y la mayoría de las cabeceras de HTTP/1.1 para facilitar su adopción.

3.6.1 SPDY

Por todo esto en el año 2011, Google comenzó a crear su propio protocolo que subsanase las deficiencias en términos de rendimiento que se habían ido encontrando a lo largo de los años en HTTP/1.1, el cual llamó SPDY.

En las pruebas de laboratorio que realizo Google obtuvo unas mejoras de entre el 40% y el 60% en los tiempos de descarga [21], tras lo cual prácticamente todos los servicios de Google fueron implementando esta tecnología. Aunque este protocolo nunca paso de la fase experimental y no llego a tener una gran acogida entre los servidores más usados sí que consiguió un gran soporte por la parte de los navegadores web, ya que los 4 navegadores principales lo soportaron rápidamente.

3.6.2 HTTP/2

En el año 2012 el IETF empezó a trabajar en un borrador de HTTP/2, el cual se basó en los principios implementados por SPDY. En el año 2015 se publicó la versión final, en la RFC7540, la cual fue rápidamente implementada por todos los navegadores web, aunque sólo sobre conexiones cifradas TLS, y empezó a ser rápidamente implementada en muchos servidores.

3.6.2.1 Navegadores que soportan HTTP/2

En la siguiente imagen se muestra en verde oscuro las versiones de los navegadores que soportan HTTP/2, en verde claro las que lo soportan si el sistema operativo es moderno (a partir de Windows 10 en el caso de IE y de OSx 10.11 en el caso de Safari) y en rojo los que no lo soportan. [22]

Actualmente en España un 95% de los navegadores usados por los internautas tienen soporte HTTP/2 y en el mundo más de un 81%.

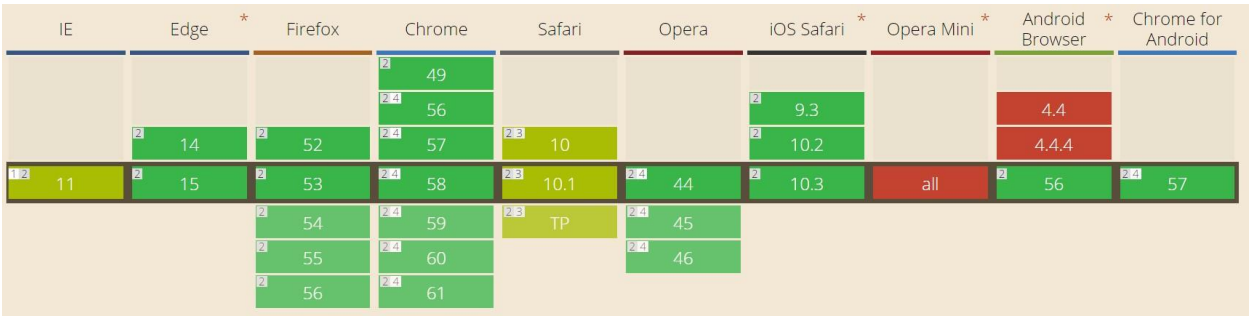


Figura 3-4 Soporte de HTTP/2 en los principales navegadores web

3.6.2.2 Mejoras implementadas en HTTP/2

El objetivo de HTTP/2 es mejorar todas las carencias de las versiones anteriores y por ello sus principales características diferenciadoras son las siguientes: [23]

- Es un protocolo binario, lo cual ahorra gran cantidad de bytes a la hora de transmitir la información respecto a HTTP/1.x que era basado en texto.
- Sólo necesita una única conexión, a diferencia de las versiones anteriores de HTTP donde era necesario

abrir múltiples conexiones TCP simultaneas para descargar varios elementos de la web en paralelo. HTTP/2 ofrece multiplexación, es decir la posibilidad de realizar múltiples peticiones y obtener los recursos en paralelo, a través de una única conexión TCP.

- Priorización de flujos. Mediante la asignación de una prioridad (entre 1 y 256) y la dependencia entre los distintos objetos, conseguimos que las tramas más urgentes para renderizar una web puedan llegar antes a su destino. Esto nos permite incrementar mucho la velocidad aparente de carga, en el caso de estar implementado correctamente en el servidor, que es quien decide que prioridad tiene cada archivo.
- Se elimina la información redundante, con lo que se intenta que no se repita el envío de datos repetidos en una misma conexión.
- Compresión de cabeceras usando HPACK, definido en el RFC7541 [24], este formato de compresión busca representar eficientemente los campos de la cabecera de HTTP comprimiéndolos y eliminando los campos redundantes en las cabeceras. Se diseñó para que fuese flexible y simple para evitar vulnerabilidades debido a malas implementaciones, como la que sufrió SPDY, que usaba DEFLATE para comprimir las cabeceras, con el ataque CRIME. [25]
- La posibilidad de enviar múltiples respuestas, en paralelo, a una única petición de un cliente, esto se ha llamado Server Push. Es muy útil cuando quieres enviar recursos adicionales como CSS o JS que el cliente va a necesitar para empezar a renderizar la web, con esto conseguimos que cuando el navegador web empiece a procesar el html y descubra que necesita un cierto archivo ya lo tendrá descargado en su cache sin necesidad de perder tiempo en hacer la petición y esperar la respuesta. [26]

Debido a las grandes mejoras de rendimiento que implican estos cambios, la mejora de seguridad al requerir cifrado y a la amplia compatibilidad que existe actualmente por parte de los navegadores webs, es necesario e importante soportar HTTP/2 cuanto antes en todas nuestras páginas web.

4 MECANISMOS DE OPTIMIZACIÓN

Good developers know how things work. Great developers know why things work.

- Ilya Grigorik -

La mejor forma para mejorar algo es empezar a mirarlo por el principio, y cuando hacemos una petición, lo primero que ocurre es la conexión TCP. La configuración del servidor puede afectar (y mucho) con lo que ocurra a partir de este momento. Por ello en este capítulo empezaré intentando explicar las mejores prácticas para que la fase de la conexión sea lo más eficiente y rápida posible. Se presupone, en la mayor parte del tema, que se están utilizando conexiones HTTPS, que son uno de los requisitos para poder utilizar HTTP/2, que tal y como se explicó en el tema anterior tiene un mayor rendimiento y es aconsejable adoptar lo antes posible. Posteriormente continuaremos viendo los distintos algoritmos de compresión y como servir los recursos adicionales a nuestros visitantes de la forma más eficiente posible.

4.1 Uso de Preconnect

Al iniciar una petición HTTP incurrimos en muchos RTTs antes de que la petición llegue al servidor, ya que el navegador tiene que resolver las DNS, realizar el *TCP handshake*, y negociar el túnel TLS si se está utilizando HTTPS.

Los navegadores modernos intentan anticiparse a las conexiones que se van a realizar para realizar toda esta preconexión antes que el usuario lo pida (resolver las DNS, negociar TCP y TLS) aunque no siempre suelen hacerlo eficientemente. A partir de Firefox 39 y de Chrome 46, podemos utilizar el *preconnect hint* [27] lo que nos permitirá indicarle al navegador que intente, siempre que pueda, resolver las DNS, iniciar una conexión TCP y negociar TLS si es necesario, y todo antes de que el navegador del usuario intente realizar la petición del recurso.

Utilizarlo es trivial, sólo tenemos que añadir en la cabecera de nuestro html la siguiente línea indicando el dominio del que vamos a necesitar el recurso instantes después:

```
<link rel="preconnect" href="//dominio.com/">
```

4.1.1 Navegadores que soportan Preconnect

En la siguiente imagen se muestra en verde las versiones de los navegadores que soportan Preconnect y en rojo los que no lo soportan. [28]

Actualmente en España más de un 74% de los navegadores usados por los internautas tienen soporte para Preconnect y en el mundo más de un 60%.

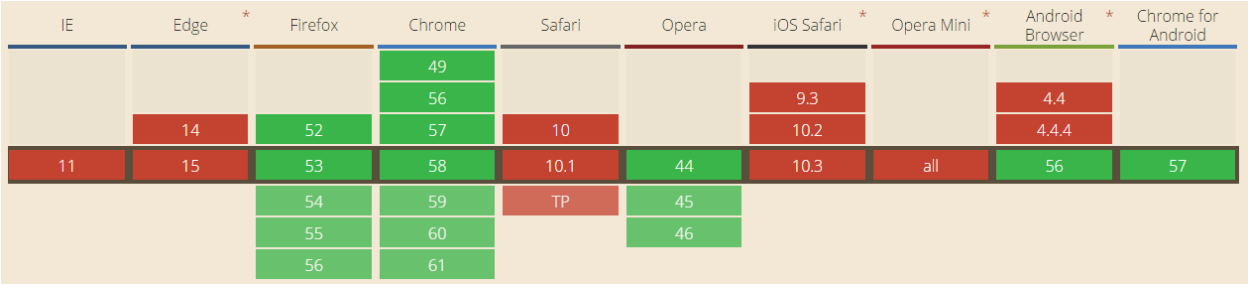


Figura 4-1 Soporte de Preconnect en los principales navegadores web

4.2 Optimizar TLS

El protocolo TLS es usado cada vez que nos conectamos a una página web con HTTPS y es especialmente importante optimizarlo en nuestros servidores lo máximo posible ya que todos los navegadores actuales sólo implementan HTTP/2 usando TLS. [29]

4.2.1 Problemas de las nuevas conexiones TLS

Uno de los problemas que podemos tener en TLS es el coste computacional del cifrado asimétrico que se produce durante la negociación TLS y que dura del orden de 1 ms como se muestra en la Tabla 4–2, mientras el cifrado simétrico que se produce al cifrar los datos de la web tiene un coste despreciable como podemos ver en la Tabla 4–1, por lo que teóricamente deberíamos reducir el número de negociaciones TLS lo máximo posible, especialmente si tenemos una web con un nivel de tráfico alto.

Para obtener las tablas siguientes hemos utilizado el siguiente comando en un servidor con procesador Intel Xenon E3-1245.

$> openssl speed sha ecdh

Tabla 4–1 Cantidad de bytes por segundo procesados con cifrado simétrico

Tipo de cifrado simétrico	1024 bytes	8192 bytes
Sha1	661.231,62K	784.220,16K
Sha256	272.628,39K	290.690,39K
Sha512	392.830,98K	450.499,93K

Tabla 4–2 Cantidad de operaciones de cifrado por segundo con cifrado asimétrico

Tipo de cifrado asimétrico	Operaciones por segundo	Duración por operación
256 bit ecdh	3323,5	0,3ms
384 bit ecdh	1557,9	0,6ms
521 bit ecdh	721,1	1,4ms

Además de esto, con la configuración por defecto, al establecer una nueva conexión TLS necesitamos usar 2 RTT sólo para establecer la conexión, y 1 RTT adicional para establecer la conexión TCP tal y como se ve en la siguiente figura.

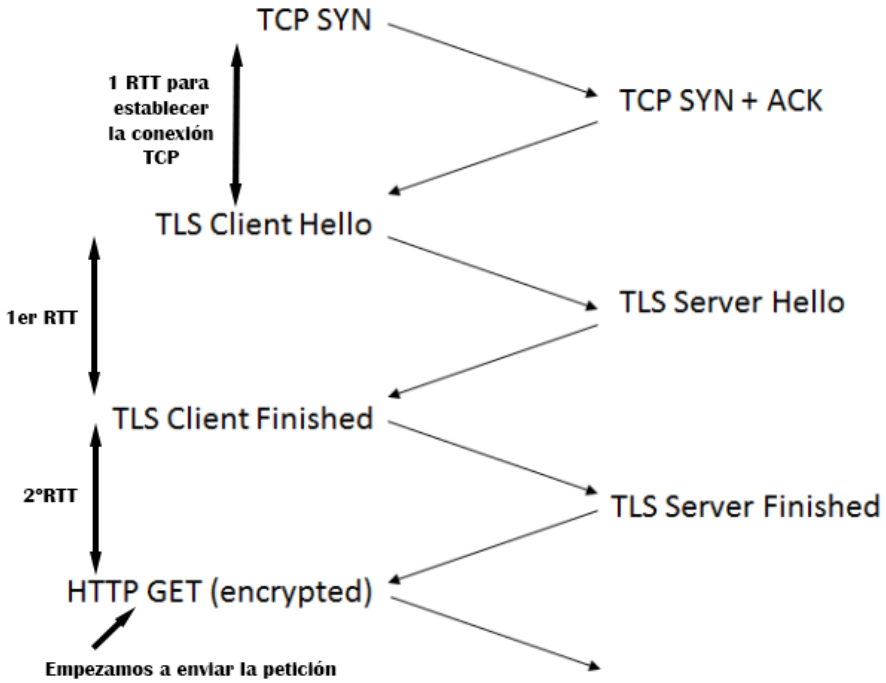


Figura 4-2 Ejemplo de conexión https indicando los RTT

4.2.2 Reutilizar las conexiones y/o sesiones TLS

La forma más sencilla de reutilizar una conexión es asegurarse que el parámetro Keep-Alive tiene un valor lo suficientemente alto como para que se pueda reutilizar para varios recursos.

Otra forma de proceder es reutilizar las sesiones usando TLS Resumption, hay dos métodos para llevarlo a cabo, y con ambos sólo necesitaremos un RTT para continuar la sesión y nos ahorraremos el costoso proceso del cifrado asimétrico que se produce durante la negociación TLS.

- El método de tickets de sesión [30], que es el menos soportado por los clientes antiguos, utiliza al cliente para guardar los datos de la sesión. Para ello el servidor cifra los parámetros de la conexión y se lo envía al cliente en forma de ticket con los datos cifrados. El cliente devuelve dicho ticket al servidor al reconectar, en ese momento el servidor lo descifra y continúa la sesión.

  Es importante revisar el timeout de los tickets, ya que por defecto en la mayoría de servidores es de 300s, lo cual es muy poco tiempo y se puede aumentar hasta un día sin problemas, tal como lo tienen la mayoría de webs importantes.

- El otro método utiliza al servidor para guardar los datos de la sesión [31], para ello el servidor asigna un identificador de sesión (ID), que envía al cliente en cada sesión TLS y guarda todos los parámetros de la sesión en una cache asociada al ID que ha enviado el servidor. El cliente al reconectar envía dicho ID, el servidor recupera los parámetros y continua la sesión. Si se utiliza este método es importante asegurarse que se tiene configurado un cache lo suficientemente grande para conservar los datos de las sesiones TLS durante un tiempo prudencial.

  $> openssl s_client -connect dominio.com:443 -tls1 -tlsextdebug -status

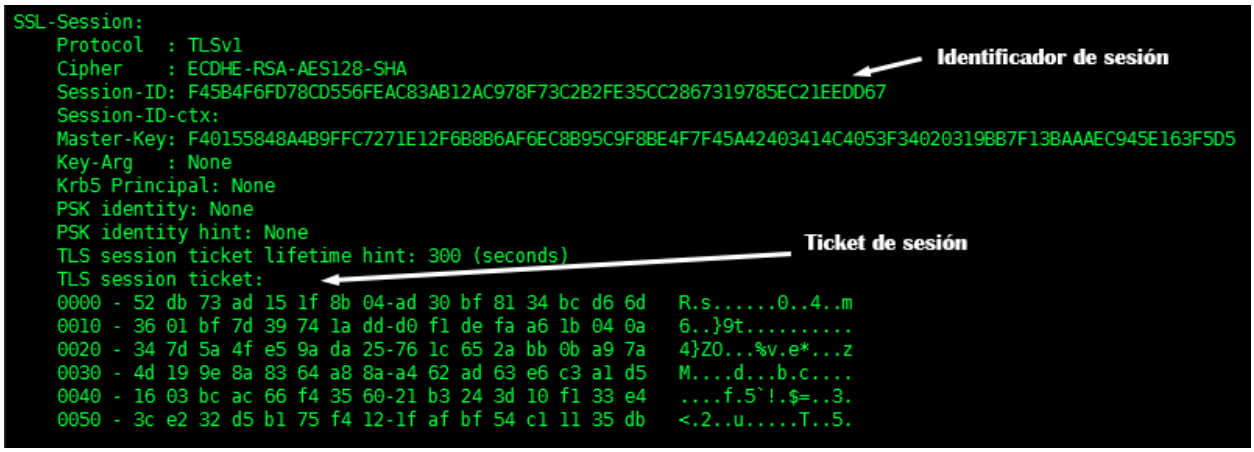


Figura 4-3 Comprobación del soporte de TLS Resumption en un Servidor Web

4.2.3 Usar TLS False Start

TLS False Start es un comportamiento opcional, de las implementaciones de los clientes, que sólo afecta a los tiempos del protocolo TLS para reducir la negociación a sólo 1 RTT en las sesiones nuevas. [32] Para ello el cliente empieza a enviar datos cifrados tan pronto como recibe el certificado del servidor y le responde que cifrado va a utilizar, sin necesidad de tener que esperar un asentimiento por parte del servidor.

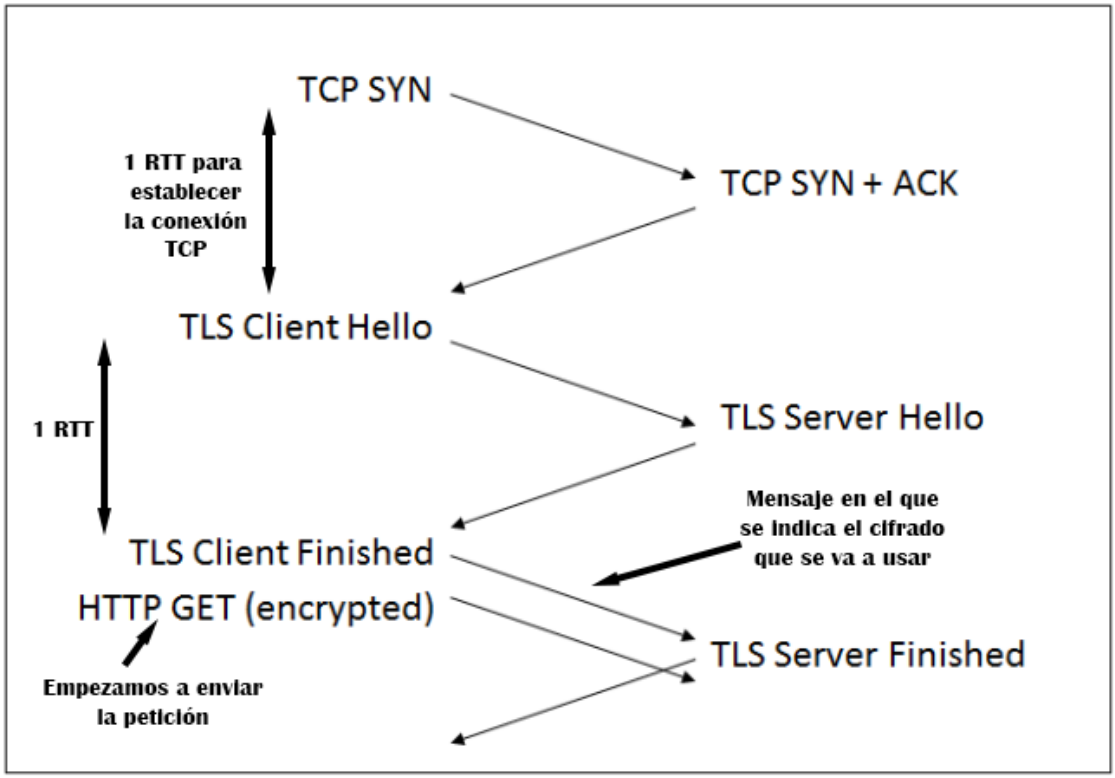


Figura 4-4 Petición HTTPS con TLS False Start

Debido a que muchos servidores antiguos y balanceadores de carga funcionan de manera incorrecta cuando un cliente intenta usar *TLS False Start*, no es algo que se intente utilizar siempre por parte de los clientes. En Chrome y Firefox comprueban previamente que el servidor utilice las extensiones NPN o ALPN [33] de TLS, que permiten negociar el protocolo de la capa de aplicación que se va a utilizar sobre TLS, antes de intentar utilizar *TLS False Start*. Por ello es importante tener dichas extensiones activas en los servidores web para así poder usar este comportamiento en los dos de los navegadores más usados actualmente y ahorrarnos un RTT al establecer una nueva sesión TLS utilizando *TLS False Start*. En las versiones más modernas de Internet Explorer se intenta utilizar siempre, y si la negociación falla se procede a reintentarlo sin usar *TLS False Start*.

4.2.4 El problema del certificado intermedio

A la hora de verificar la validez del certificado de un dominio, se comprueba usando el certificado intermedio de la entidad certificadora (CA), si no proporcionamos dicho certificado intermedio el cliente puede pausar la carga de la web, mientras obtiene dicho certificado. Los certificados intermedios se comprueban usando los certificados raíz, el cual no es aconsejable añadir ya que aumenta el tamaño de lo que tenemos que enviar y no aporta nada, debido a que los navegadores ya los incluyen y en el caso de que no lo hiciesen no se fiarían de uno que le enviásemos.

Podemos crear el archivo que incluya nuestro certificado y el certificado intermedio fácilmente utilizando el terminal de Linux:

$> cat midominio.cert intermedio.cert > completo.cert

4.2.5 Comprobaciones OCSP (Online Certificate Status Protocol)

Algunos navegadores comprueban que el certificado no este revocado, esto añade un problema adicional debido a que pausa la carga antes de que realmente empiece, mientras el navegador resuelve las DNS, negocia la conexión TCP y envía la petición HTTP a la entidad certificadora.

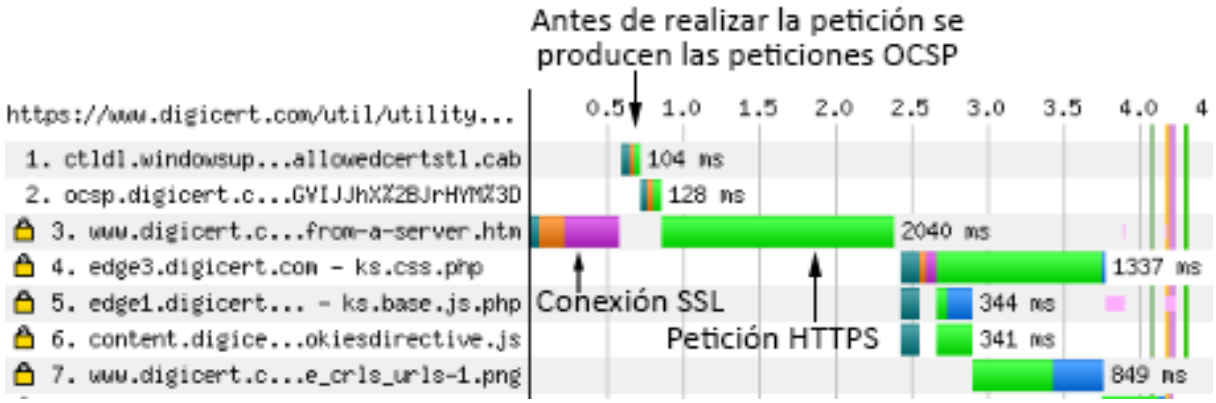


Figura 4-5 Petición HTTPS interrumpida por OCSP

La solución a este problema es utilizar OCSP Stapling [34], que hace que el servidor compruebe previamente la validez del certificado e informe al cliente enviándole la respuesta firmada por la entidad certificadora, con ello reducimos el tiempo necesario para la comprobación.

Podemos comprobar si nuestro servidor utiliza OCSP Stapling usando el siguiente comando y comprobando que en la respuesta obtenemos datos de OCSP:

$> openssl s_client -connect miservidor.com:443 -tls1 -tlsextdebug -status

4.2.6 Tener cuidado con las redirecciones

Al utilizar HTTPS es común utilizar redirecciones desde HTTP, pero si no tenemos cuidado podemos llegar a encadenar múltiples redirecciones. Las redirecciones al usar HTTPS son aún más costosas debido al RTT extra que conlleva la negociación. Y en el caso de conexiones móviles que tienen una latencia mayor, puede conllevar más de 1 segundo extra.

Por ello debemos evitar redirecciones en cadena del siguiente tipo, que son bastante comunes:

- Enviar del protocolo HTTP a la url con el protocolo HTTPS, de la url sin www a la url con www, y posteriormente comprobamos si es un móvil, y si lo es reenviamos a la versión móvil (que tiene otra url diferente).

La manera correcta de hacerlo sería la siguiente:

- Si entramos a través del protocolo HTTP, comprobamos si es un móvil y en ese caso lo enviamos directamente a la url de la versión móvil usando el protocolo HTTPS. En caso de que no sea un móvil enviamos directamente a la versión con www y protocolo HTTPS. De esta forma sólo necesitamos una sola redirección ahorrando mucho tiempo a nuestro usuario.

4.2.7 Implementar HSTS si sólo utilizas HTTPS

Mediante el mecanismo de HSTS [35] las páginas webs pueden declarar, a través de la cabecera de respuesta de HTTP, que sólo utilizan HTTPS. De esta forma el navegador, a partir de entonces, redireccionará automáticamente cualquier intento de acceder al dominio utilizando HTTP a la versión HTTPS sin necesidad de hacer ninguna petición.

4.2.8 Optimizar el tamaño de los paquetes TLS dinámicamente

El protocolo TLS cifra los datos de la capa de aplicación en paquetes de tamaño variable. Nos interesa que ese tamaño inicialmente sea menor que la cantidad de datos que podemos enviar en 1 RTT, lo cual depende de la ventana de congestión del cliente, debido a que, si intentamos enviar paquetes de 16KB, el máximo permitido por TLS, normalmente el cliente tendrá que esperar hasta el siguiente RTT para empezar a procesar los datos que ha recibido, ya que no se puede descifrar un paquete incompleto.

Por ello la configuración ideal es que las nuevas conexiones empiecen con paquetes de 1400B, que es el tamaño de 1 MTU, después de que cierta cantidad de datos haya sido enviada se suba el tamaño de los paquetes a 16KB y en caso de inactividad mayor a 1s se vuelva a los paquetes de 1400b. Actualmente Apache y cada vez más servidores permiten el ajuste dinámico del tamaño de los paquetes TLS.

4.3 Compresión

Todos los recursos basados en texto, como el HTML, CSS o JavaScript, pueden beneficiarse de los beneficios de la compresión. Los navegadores web indican el soporte de las diferentes formas de compresión a través de la cabecera *Accept-Encoding* de HTTP y los servidores indican la forma de compresión utilizada a través de la cabecera *Content-Encoding*.

4.3.1 DEFLATE y Gzip

Estas dos son actualmente las formas de compresión más utilizadas. En HTTP/1.1 [36] cuando nos referimos a DEFLATE realmente nos estamos refiriendo al formato Zlib, que consta de datos comprimidos con el mecanismo de compresión deflate junto a un checksum ADLER32. Gzip también utiliza el mecanismo de compresión deflate, pero junto a un checksum CRC-32.

Actualmente, con los procesadores multinucleos, la velocidad de compresión y descompresión de las dos es muy similar, aunque DEFLATE es ligeramente más rápido debido a que ADLER32 está diseñado para ser más eficiente que CRC-32.

4.3.2 Brotli

Esta nueva forma de compresión, creada por empleados de Google, fue lanzada en 2015 para mejorar la compresión de tipografías web, ya que es el utilizada en el popular formato WOFF2, pero actualmente se puede utilizar también sobre cualquier otro tipo de recurso y es una RFC del IETF desde Julio de 2016. [37]

Se pueden llegar a obtener mejoras de compresión superiores al 25% respecto a deflate [38] además de mejoras en la velocidad de descompresión. La compresión es notablemente más lenta, por lo que en una página web es aconsejable tener los recursos estáticos previamente comprimidos para no tener que realizar la compresión en tiempo real.

Debido a los problemas que presentan muchos servidores proxies, al ser incapaz de gestionar adecuadamente contenido comprimido de una forma diferente a deflate o gzip, los navegadores webs sólo implementan Brotli

cuando se utiliza HTTPS.

4.3.2.1 Navegadores que soportan Brotli

En la siguiente imagen se muestra en verde las versiones de los navegadores que actualmente soportan recibir datos comprimidos con Brotli y en rojo los que no lo soportan. [39]

Actualmente en España más de un 72% de los navegadores usados por los internautas tienen soporte para usar Brotli y en el mundo más de un 58%.

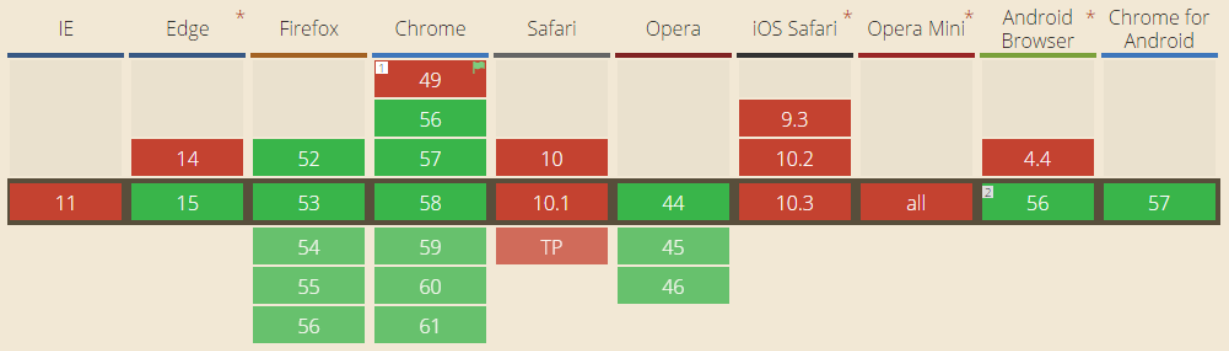


Figura 4-6 Soporte de Brotli en los principales navegadores web

4.4 Imágenes

Actualmente las imágenes son el recurso que aglutina la mayor parte del tamaño de una web. Elegir el formato de cada una y optimizarlas es una tarea muy importante, ya que puede ahorrarnos muchos kilobytes y acelerar la carga de nuestra web.

Los tipos de imágenes clásicos son:

- PNG-8: Perfecta para utilizar en imágenes con pocos colores.
- PNG-24: Es el formato que debemos usar cuando tenenos una imagen con transparencias.
- JPG: Se debe usar para fotografias y para imágenes complejas o con muchos colores.
- GIF: No es aconsejable usarlos en la actualidad, excepto para archivos con muy pocos pixeles o con menos de 3 colores. Originalmente se aconsejaba utilizarlos en los casos donde necesitásemos imágenes con movimiento, pero actualmente es preferible utilizar el formato mp4.

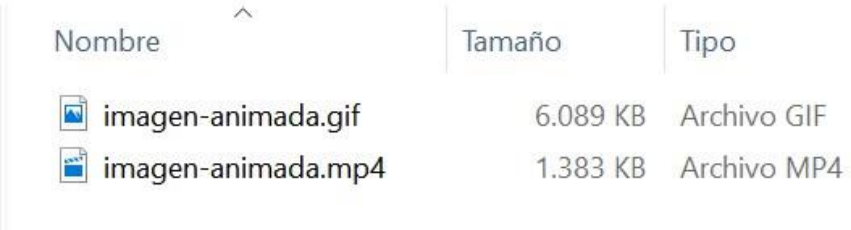


Figura 4-7 Tamaño de una misma imagen con movimiento en gif y en mp4

Para introducir un recurso mp4 y que en los navegadores web de nuestros visitantes se vea igual que los gifs, utilizaremos el siguiente código adaptando el tamaño de los atributos width y height al tamaño con el que queramos que se muestre.

```
<video autoplay="autoplay" loop="loop" muted="" width="300" height="300"><source src="la-url-del-video-que-quieras-poner-en-mp4" type="video/mp4" /><video>
```

4.4.1 WebP

Este nuevo formato creado por Google en 2010 está pensado para sustituir a las imágenes JPG y PNG debido a que ofrece de media mejoras de tamaño superiores al 25%, disminuyendo los tiempos de decodificación de la imagen y sin un aumento significativo en el consumo de RAM.

La parte negativa de WebP es que actualmente solo está soportado por Chrome y Opera, y aunque Safari y Firefox empezaron a implementarlo a finales de 2016, Microsoft ha declarado que no va a implementarlo en sus navegadores. Debido a esto, para utilizar WebP necesitamos asegurarnos que el navegador de nuestro visitante puede utilizar el formato WebP antes de servírselo, esto podemos hacerlo en el servidor comprobando que el cliente nos ha enviado la cabecera indicando que acepta "image/webp" o en el propio navegador utilizando JavaScript.

4.4.2 Optimizar imágenes

Las imágenes JPG y PNG se pueden optimizar de multiples formas, como reduciendo la profundidad del color al nivel visible en una pantalla, eliminando datos adicionales como la fecha en la que se creo o los detalles de la cámara con la que se hizo la fotografía.

Es importante utilizar el tamaño de imagen máximo con el que se vaya a visualizar posteriormente en el navegador, para evitar descargar imágenes más grandes de lo necesario y que el navegador tenga que perder tiempo escalándolas durante el renderizado.

Además, también se pueden comprimir sin perdida de calidad, y en el caso de las imágenes JPG es posible comprimirlas aún más con algo de perdida de calidad. Los compresores más utilizados son jpegtran y jpegoptim para JPG y OptiPNG y PNGOUT para PNG

4.4.3 Srcset

Fue una de las mejoras que llego con HTML5 y que actualmente soportan todos los navegadores modernos, exceptuando Internet Explorer y Edge.

Este atributo nos permite definir distintos tamaños de una misma imagen para que el navegador decida por si mismo cual es el tamaño que necesita utilizar, lo cual nos es muy útil si tenemos un diseño en el que el tamaño de las imágenes varía con el tamaño de la pantalla. También se puede utilizar para proporcionar imágenes el doble de grandes para una correcta visualización en las pantallas retina donde el tamaño de un pixel CSS equivale a dos pixeles reales. En el caso de agregar también el atributo src, este será usado cuando el navegador no sea compatible con srcset.

Utilizarlo es muy sencillo, sólo debemos proporcionar cada tamaño de la imagen seguido del ancho de la imagen en pixeles seguido de "w" tal y como se muestra en el código siguiente:

```
<img src="imagen-reponsive.jpg" srcset=" imagen-reponsive-160.jpg 160w, imagen-reponsive-320.jpg
320w, imagen-reponsive-640.jpg 640w, imagen-reponsive-1280.jpg 1280w">
```

4.5 CSS

En cuanto al CSS es importante optimizar la forma en que lo servimos al navegador, debido a que para obtener un renderizado rápido de la parte superior de la página hay que evitar incluir muchos recursos CSS externos en la cabecera de la misma, ya que bloquean el renderizado, por lo que al menos debemos minimizar su uso en este lugar.

Lo ideal es identificar los estilos críticos para el diseño de la parte superior de la página y cargarlos junto al html mediante las etiquetas <*style*></*style*>, y retrasar la carga del resto de los recursos CSS después del html necesario para mostrar la parte superior de la página.

También es importante minificar los recursos CSS adicionales quitándoles todos los espacios en blanco, saltos de línea y tabuladores posibles; de esta forma aceleramos la descarga y el análisis. Y si estamos utilizando HTTP/1.X debemos intentar minimizar la cantidad de recursos adicionales CSS que el navegador tenga que

descargar.

Además, hay que evitar utilizar estilos CSS *inline* en el html, mediante el atributo style, ya que los navegadores tardan más tiempo en procesar y renderizar dichos estilos frente a los otros dos métodos. Adicionalmente hay que tener en cuenta que el uso de este tipo de estilos complica el mantenimiento de una página web.

4.6 JavaScript

El código JavaScript es el responsable de muchos efectos y animaciones que podemos ver prácticamente en todas las páginas webs. El uso de JavaScript requiere una capacidad de procesamiento adicional, lo cual penaliza, especialmente en entornos móviles, a las webs que lo usan en exceso.

La forma y el momento en que cargamos el código JavaScript pueden afectar enormemente al tiempo requerido hasta el primer renderizado y con ello a la velocidad de carga percibida por nuestros visitantes.

La carga, por defecto, de recursos JavaScript externos se lleva a cabo de forma que bloquea el renderizado y la visualización del contenido, mientras se descargan y procesan. Por ello es importante tomar las siguientes medidas:

- Poner siempre los recursos JavaScript tras todos los archivos y código CSS
- Poner en el encabezado el mínimo posible de Javascript, que sea realmente necesario para la carga inicial de la página, y siempre que sea posible incluirlo directamente con las etiquetas *<script type="text/javascript"></script>* en vez de como un archivo externo.
- Cargar asíncronamente los recursos que no sean vitales para la página, de esta forma evitamos que bloqueen el renderizado. Podemos hacerlo añadiendo el atributo *async* de la siguiente forma:

```
<script async src="archivo.js">
```

- Minificar los recursos JavaScript para acelerar la descarga, análisis y tiempo de ejecución.

En el caso de que estemos utilizando HTTP/1.X también es útil agrupar varios recursos JavaScript en un solo archivo para evitar la penalización que implica abrir múltiples conexiones con el servidor para descargarlos.

4.7 Fuentes Web

Actualmente para usar una fuente diferente a las disponibles del sistema operativo, sólo hay que utilizar la propiedad @font-face de CSS. El principal problema ocurre con la forma en que los distintos navegadores la gestionan, además hay que tener en cuenta que puede implicar un sobrecoste en el tamaño total de la página.

La propiedad @font-face permite definir varios formatos de archivo y que cada navegador escoja el primero que pueda utilizar. Con el soporte actual, sólo es necesario incluir los archivos de fuente usando el formato WOFF y WOFF2 (que tiene tamaño un 30% menor). Es importante que el archivo WOFF2 se encuentre antes que el archivo WOFF, debido a que en caso contrario nunca se usaría.

El resto de formatos tienen un tamaño mucho mayor y sólo son necesarios para dar soporte a IE6, IE7 e IE8, por lo que su uso no está aconsejado.

```
/* Definimos nuestra nueva fuente */
@font-face {
        font-family: 'MiNuevaFuente';
        src: url('mifuente-Normal.woff2') format("woff2"),
        url('mifuente-Normal.woff') format("woff");
}
```

El mayor problema de las fuentes web es el llamado FOIT (*Flash Of Invisible Text*), que es debido a que la mayoría de los navegadores no muestran texto alguno mientras están descargando y procesando el archivo de la fuente. El comportamiento habitual, en la actualidad, es el de esperar 3 segundos sin mostrar nada antes de mostrar la fuente por defecto, y en caso de que la fuente web llegue a estar disponible se cambia por la fuente por defecto.

Lo ideal en el caso de las fuentes, especialmente ahora con tantos dispositivos móviles que tienen conexiones más lentas, es que ocurriese el llamado FOUT (*Flash Of Unstyled Text*) que implicaría que se mostrase el texto en el momento del renderizado, pero usando una fuente del sistema, y cuando nuestra fuente se haya decargado y procesado que sustituya a la fuente usada por defecto. Esto nos permite que nuestros usuarios puedan navegar lo más rápido posible por nuestra web, lo cual nos ayudará a retener una mayor cantidad de usuarios al evitar que abandonen la web al ver las imágenes y el diseño, pero sin poder leer el texto.

4.7.1 Font-display

La propiedad CSS font-display será la solución más sencilla y rápida en cuanto se empiece a implementar en los distintos navegadores webs, actualmente ha tenido una gran aceptación y es parte del borrador de CSS Fonts 4 [40]. Desde mediados del mes de Junio de 2017 está implementada en las versiones de prueba de Chrome [41] por lo que es de esperar que en unos meses podamos empezar a usarla.

Esta propiedad, que se pondrá en el interior de cada @font-face, permitirá que los diseñadores puedan decidir la forma en que cargue cada una de las fuentes según sus intereses, y sin necesidad de tener que utilizar JavaScript.

Los valores que se le podrá dar a está propiedad serán:

- *auto*, utiliza el comportamiento por defecto del navegador, actualmente es igual a *block*.
- *block*, espera a que se descargue el archivo de fuente durante 3 segundos (periodo de block) sin mostrar nada y posteriormente muestra la fuente del sistema correspondiente hasta que se consiga descargar y procesar, momento en que se cambian.
- *swap*, muestra la fuente del sistema en el momento del renderizado y la cambia por la fuente web en cuanto este disponible.
- *fallback*, espera un pequeño periodo de block de 100ms y posteriormente tiene 3 segundos de periodo de swap tras el cual se quedará la fuente del sistema si no se ha conseguido descargar y procesar la fuente web.
- *optional,* espera un pequeño periodo de block de 100ms y si la fuente no está lista, se usará y quedará la fuente del sistema. Posteriormente la descarga de la fuente puede continuar con una prioridad baja para usarla en futuras visitas, o se puede cancelar la descarga si la conexión es mala.

5 ANALISIS Y OPTIMIZACIÓN

All communication should be secure, always, and by default! HTTPS everywhere!

- Ilya Grigorik -

Para esta demostración, vamos a optimizar la página web apocalipsisz.es que está basada en Wordpress, ya que actualmente es el gestor de contenidos (*CMS*) más usado en el mundo. El servidor donde se encuentra alojada la página web tiene un procesador Intel E3-1225v2, 32GB de memoria RAM, discos duros SSD y 250Mbps de ancho de banda disponible, por lo que en ningún momento ninguno de los componentes del servidor debería hacer cuello de botella afectando a las mediciones. El programa servidor utilizado en todas las mediciones es Apache 2.4.

Vamos a medir primero la página sin optimizar, pero con distintas versiones de PHP para medir si existen mejoras de rendimiento apreciables entre ellas en un wordpress.

Posteriormente vamos a realizar mediciones tras mejorar sólo algo en concreto para poder valorar que mejoras tienen un mayor impacto.

Para finalizar con todo lo aprendido iremos realizando mejoras acumulativas hasta obtener una página web completamente optimizada. Ninguno de los cambios realizados durante las optimizaciones afectará al diseño o a las funcionalidades de la página web.

5.1 Metodología de las mediciones

El informe de Akamai del estado de internet en el primer cuatrimestre de 2017 [42] afirma sobre las conexiones en España que:

- La velocidad media de las conexiones fijas (no móviles) es de 15,5Mbps
- El 90% de la población con internet tiene una conexión fija de más de 4Mbps
- El 56% de la población con internet tiene una conexión fija de más de 10Mbps
- El 36% de la población con internet tiene una conexión fija de más de 15Mbps
- La velocidad media de las conexiones móviles en España es de 13,8Mbps

A partir de estos datos, del informe del ministerio [43] y del informe de calidad de la red en Europa de SamKnows [44] he modelado 4 conexiones tipo:

- Una conexión basada en ADSL con 8,5Mbps de bajada y 0,7 de subida, y una latencia de 60ms
- Una conexión basada en la velocidad media con 15,5Mbps de bajada y 1,5Mbps de subida, y una latencia de 45ms
- Una conexión basada en FTTH con 50Mbps de bajada y 5Mbps de subida, y una latencia de 20ms
- Una conexión móvil basada en HSPA+ con 10Mbps de bajada y 3Mbps de subida y una latencia de 100ms

Para realizar las mediciones usaremos la instancia privada de WebPageTest que hemos montado *(Consultar el Anexo A)* junto a un script escrito en NodeJS que hace las peticiones a la API de WebPageTest y organiza los datos *(Consultar el Anexo B)*.

Para cada caso realizaremos 20 medidas por cada una de las conexiones modeladas anteriormente y calcularemos la media de cada métrica por conexión. En el caso de la conexión móvil utilizaremos emulación de escritorio para imitar el comportamiento de un móvil Android con Chrome, mientras en el resto de casos utilizaremos la versión Chrome 59 para Windows.

Las métricas que utilizaremos para analizar los resultados son algunas de las comentadas en el apartado 3.1, en concreto empezaremos teniendo en cuenta el tiempo del primer byte en el apartado 5.2 ya que estamos midiendo tiempos que ocurren en el backend y es la medida que tiene más sentido comparar en estos casos.

A continuación, nos centraremos en mejorar el *Speed Index* ya que es la métrica que mejor se adapta a las sensaciones que tienen nuestros visitantes sobre la veocidad de nuestra página web. En algunos casos compararemos también el tamaño total de la suma de todos los recursos que forman la página web o la velocidad de carga completa de la web, ya que son medidas muy interesantes para complementar al *Speed Index*

5.2 Diferencias de rendimiento entre distintas versiones de PHP

Hemos comparado las distintas versiones de PHP que están disponibles en la mayoría de proveedores de Hosting actualmente.

Para empezar, mediremos el tiempo del primer byte (TTFB) utilizando HTTP/1.1 y sin usar HTTPS, para poder comparar los distintos tiempos de procesamiento:

Tabla 5–1 Tiempo del primer Byte sin HTTPS usando distintas versiones de PHP

Version\Conexión	*HSPA+*	*ADSL*	*MEDIA*	*FTTH*
PHP 5.4	613 ms	499 ms	455 ms	403 ms
PHP 5.5	654 ms	501 ms	452 ms	403 ms
PHP 5.6	607 ms	496 ms	446 ms	403 ms
PHP 7.0	567 ms	447 ms	400 ms	356 ms
PHP 7.1	554 ms	473 ms	401 ms	345 ms

Posteriormente comparamos las mismas versiones utilizando HTTPS sobre HTTP/2 utilizando las optimizaciones TLS que indicamos en el apartado 4.2. Como podemos ver en la Tabla 5–2 hay un pequeño aumento que aparece aproximadamente constante en cada conexión, el cual es debido al RTT extra que implica establecer la sesión TLS. Aunque en un principio nos pueda parecer perjudicial nos proporciona una conexión segura y debido al resto de mejoras de HTTP/2, la carga de la página completa termina siendo ligeramente más rápida.

Tabla 5–2 Tiempo del primer Byte con HTTPS usando distintas versiones de PHP

Version\Conexión	*HSPA+*	*ADSL*	*MEDIA*	*FTTH*
PHP 5.4	774 ms	630 ms	574 ms	471 ms
PHP 5.5	782 ms	627 ms	561 ms	471 ms
PHP 5.6	768 ms	622 ms	558 ms	463 ms
PHP 7.0	712 ms	578 ms	502 ms	400 ms
PHP 7.1	717 ms	570 ms	496 ms	409 ms

Despues de ver los resultados obtenidos, podemos decir sin lugar a duda que las últimas versiones de PHP 7.X tienen un rendimiento sustancialmente mejor que las antiguas versiones PHP 5.X ya que en todos los casos hemos obtenido un tiempo del primer byte más rápido, con un margen de entre 45 y 78 ms.

5.3 Optimizaciones individuales

En todas las mediciones a partir de este momento vamos a utilizar PHP 7.1 y también usaremos HTTPS con HTTP/2 ya que en el apartado anterior hemos observado que es una de las dos versiones con las que se obtienen mejores resultados.

5.3.1 Optimizar imágenes y adjustarlas al tamaño con el que se usan

Vamos a dividir las pruebas de este apartado en varias partes: optimización de todas las imágenes sin perdida de calidad, optimización de las imágenes con perdida de calidad no visible al ojo humano en una pantalla y ajuste del tamaño de las imágenes al tamaño en el que realmente se usan y con perdida no visible.

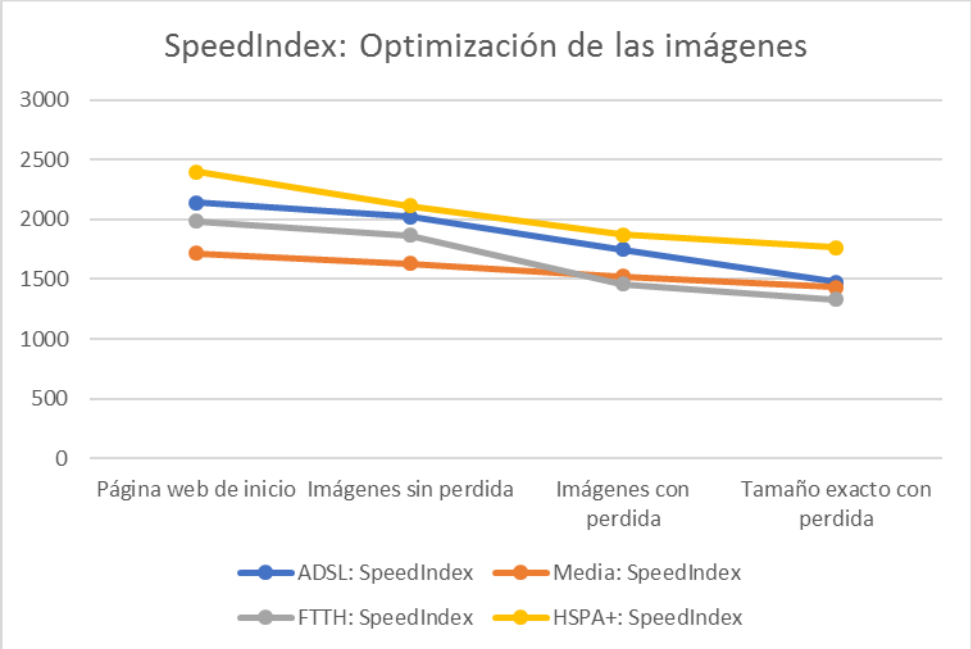


Figura 5-1 Medida del SpeedIndex usando varios métodos de optimización de imágenes

Cómo podemos ver en los resultados obtenidos la mejora es evidente, consiguiendo mejoras de hasta un 35% en el Speed Index y mejoras de entre un 26% y un 47% en el Tiempo de carga completa, al optimizar todas las imágenes y adaptar al tamaño adecuado las imágenes con un tamaño mayor al utilizado en la página web.

Para la optimización de las imágenes con perdida de calidad, lo cúal sólo ocurre en las imágenes con formato jpg, hemos utilizado la opción "Lossy" de la herramienta Kraken [45] que mediante múltiples técnicas utiliza el factor de calidad mínimo posible para que el ojo humano, en prácticamente todos los casos, no pueda distinguir la imagen optimizada de la original al verla en una pantalla. Este tipo de compresión es de utilidad para la gran mayoría de las páginas webs actuales, donde las imágenes son sólo un añadido visual.

Otra de las grandes ventajas de la optimización de las imágenes es el gran ahorro en ancho de banda que supone, lo cúal es muy útil para cuando nuestros visitantes nos visitan desde un teléfono móvil ya que disponen de planes con un consumo limitado. Además, las páginas webs con muchos visitantes pueden ahorrar mucho dinero en ancho de banda, debido a que la reducción del tamaño habitualmente supera con creces el 50% respecto al tamaño de la página web con imágenes sin optimizar.

A continuación, tenemos el tamaño en KB de la página web tras cada tipo de optimización:

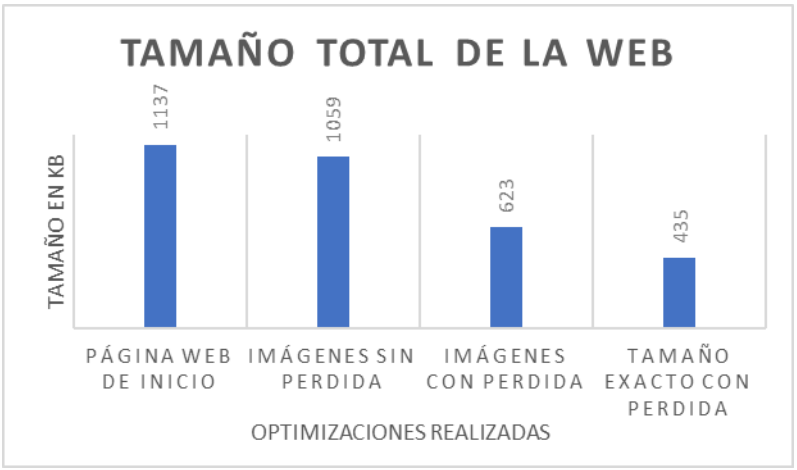


Figura 5-2 Tamaño total de la página web con distintos métodos de optimización de imágenes

5.3.2 Minificar los archivos JavaScript y CSS

En esta prueba hemos minificado todos los recursos JavaScript y CSS como se indica en el apartado 4.5 y 4.6 y hemos procedido a utilizar JavaScript asíncrono utilizando la propiedad "*async*"

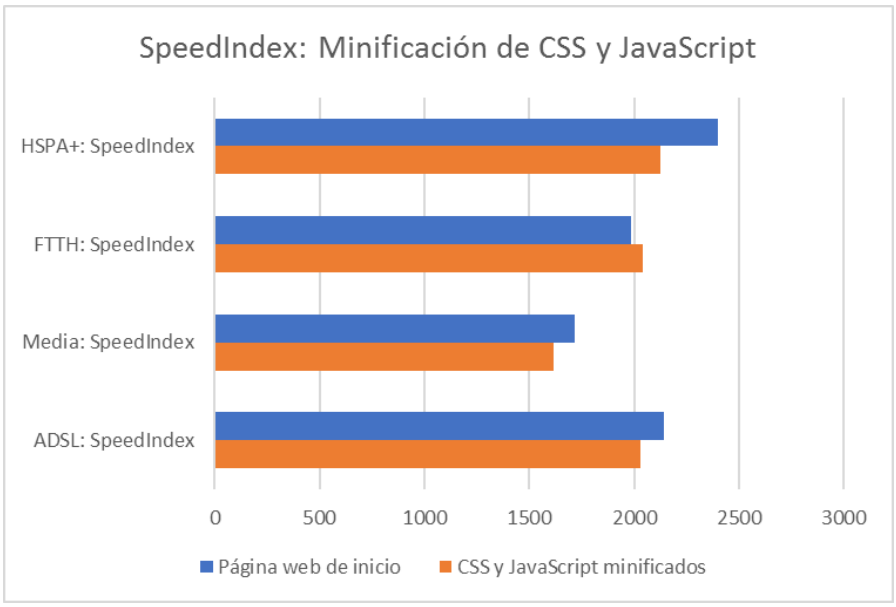


Figura 5-3 Medida del SpeedIndex al minificar CSS y JavaScript

Podemos observar que la minificación resulta en una mejora del SpeedIndex, excepto para el caso de la conexión con fibra óptica, aunque en este caso se observa que el tiempo total de carga es ligeramente inferior como vemos en la siguiente gráfica:

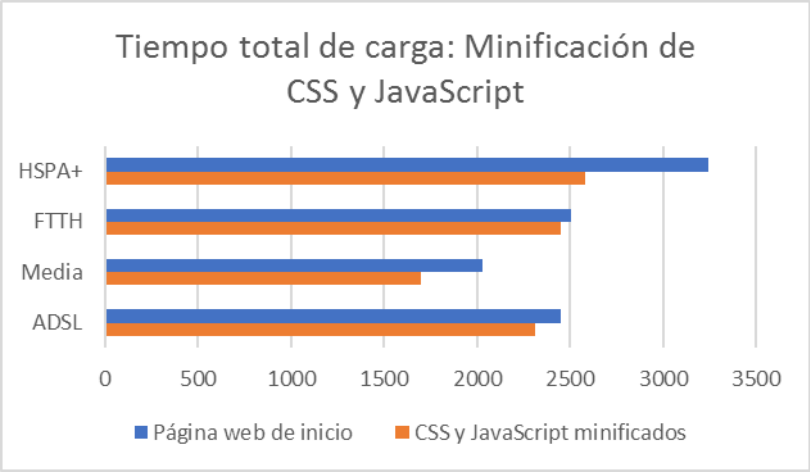


Figura 5-4 Tiempo total de carga al minificar CSS y JavaScript

Por otro lado, vemos una pequeña reducción en el tamaño total de la web, gracias al menor tamaño de los recursos CSS y JavaScript. Esto afecta muy positivamente a entornos con menor velocidad y mayor latencia como son los dispositivos móviles por eso observamos una reducción de medio segundo en nuestra conexión HSPA+

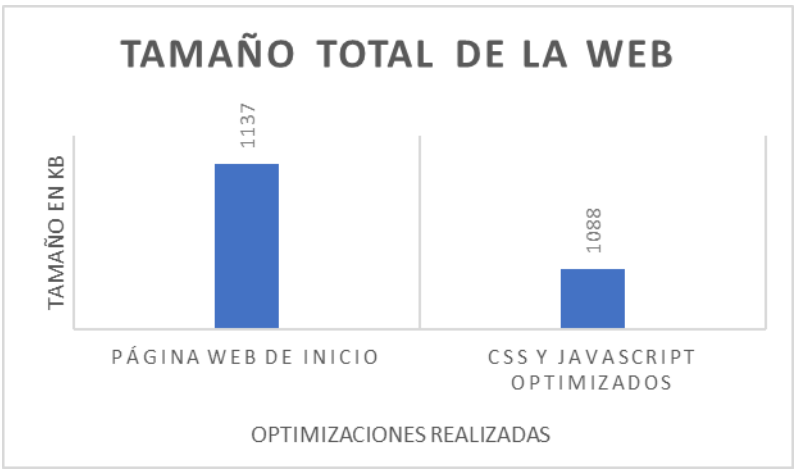


Figura 5-5 Tamaño total de la página web al minificar CSS y JavaScript

5.3.3 Habilitar la compresión

Hemos empezado activando la compresión DEFLATE en todos los recursos, y para continuar hemos experimentado usando el nuevo algoritmo BROTLI, el cual ha sido un poco más complejo de aplicar debido a la falta de programas e instrucciones. Por ello, tras investigar, lo hemos implementado precomprimiendo los recursos, usando la herramienta creada por Google (Explicada en el Anexo C) y posteriormente enviando dicha versión de los recursos a los usuarios, con navegadores que aceptan este algoritmo, gracias a la flexibilidad que nos ofrece mod_rewrite de Apache.

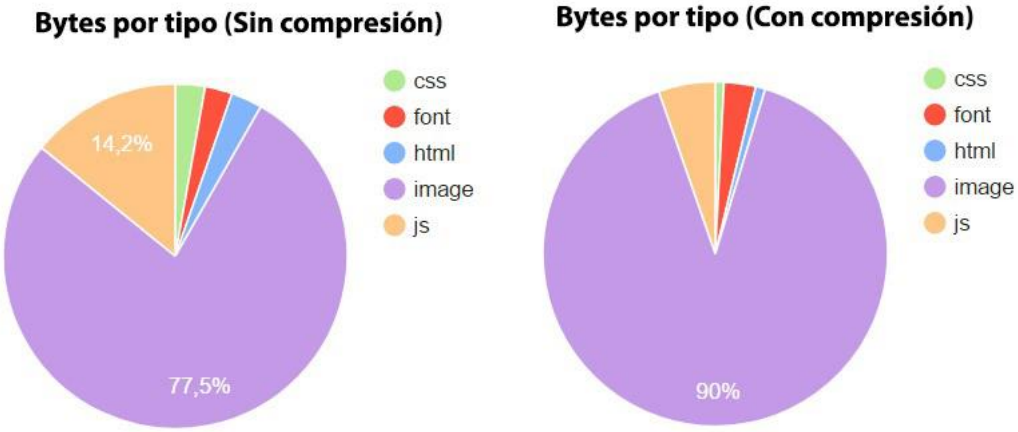


Figura 5-6 Comparación del porcentaje de bytes por tipo de archivo antes y después de la compresión

Las imágenes no pueden ser comprimidas aún más ya que cada formato tiene su propia forma de optimización interna como vimos en 4.4.2 y las fuentes ya llevan la compresión en Brotli incluida en su formato (WOFF2) por eso son los únicos recursos que el tamaño transmitido es ligeramente superior debido a las cabeceras tal y como podemos ver en la tabla Tabla 5–3 Tamaño en Bytes transmitidos de los recursos comprimidos y su tamaño sin comprimir.

Tabla 5–3 Tamaño en Bytes transmitidos de los recursos comprimidos y su tamaño sin comprimir

Tipo MIME	*Bytes transmitidos*	*Bytes originales*
Imágenes	855.205	852.303
Js	50.573	155.586
Fuentes	28.148	27.812
HTML	8.555	32.140
CSS	7.723	35.784

5.3.4 Enviar cabeceras de cache al navegador

Al configurar nuestro servidor para que envíe cabeceras de cache al navegador de los visitantes conseguimos hacer que en las próximas visitas, o incluso si visitase otra página de nuestra página web, podamos reducir el tiempo de carga ampliamente ya que el navegador web no tiene que volver a descargar los archivos que ya había descargado previamente.

A continuación, se muestra la amplia diferencia en el tiempo de carga completa de una primera carga comparada con una carga posterior habiendo enviado las cabeceras de cache correctamente.

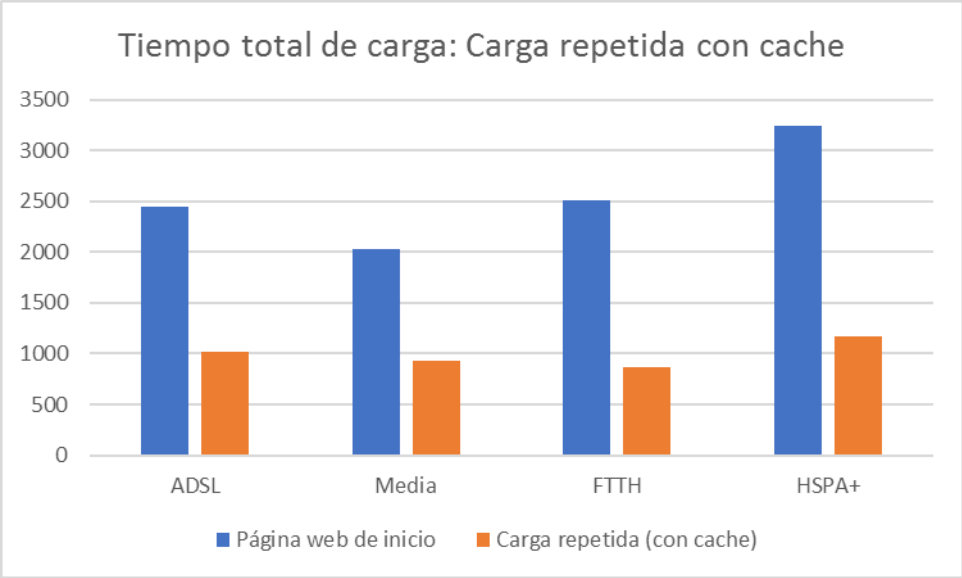


Figura 5-7 Comparación del tiempo total de carga usando cache en el navegador

En las últimas versiones de Google Chrome el navegador guarda en cache todo lo que recibe, de forma temporal, en caso de que no haya recibido ningún tipo de indicaciones sobre como gestionar el cache. Esto no ocurre en otros navegadores o versiones más antiguas de Google Chrome y por ello es importante no olvidarnos de hacer este cambio.

5.4 Optimización completa

Para terminar, vamos a realizar una optimización completa de la misma página web. La configuración de partida es Apache 2.4 con PHP 7.1 y HTTPS usando el protocolo HTTP/2.

El objetivo es conseguir que la página web quede lo mejor optimizada posible combinando las optimizaciones que han dado mejores resultados en la sección 5.3, y que no sean conflictivas entre ellas.

Antes de empezar hemos realizado un análisis de la página en la herramienta GTmetrix [7] que nos ofrece datos interesantes como la puntuación de PageSpeed y de YSlow, las cuales explicamos en la sección 3.5. Cómo podemos ver la puntuación es muy baja ya que una página web optimizada siempre debería pasar de 90% en PageSpeed y de 80% en YSlow, ya que tenemos que tener en cuenta que algunas de las recomendaciones ofrecidas pueden ser contraproducentes o imposibles de llevar a cabo debido a que dependan de servicios externos.

Figura 5-8 Análisis de la página usando GTmetrix antes de la optimización

Hemos empezado ajustando el tamaño de las imágenes al tamaño con el que se utilizan en la página web y optimizando todas las imágenes con pérdida de calidad no visibles al ojo humano a través de una pantalla. Tras esto, hemos minificado todos los archivos JavaScript y CSS, y posteriormente activado la compresión de los recursos de forma que si el navegador visitante acepta BROTLI se le envíen los recursos comprimidos en este formato y en caso contrario se le envíen comprimidos con DEFLATE.

Debido al mecanismo que hemos utilizado para comprimir en BROTLI y a que el código HTML se genera de forma dinámica, sólo hemos podido comprimir el HTML usando DEFLATE para las pruebas.

Tabla 5–4 Tiempo de carga completa tras las optimizaciones

	ADSL	*% respecto a inicio*	***Media***	*% respecto a inicio*	***FTTH***	*% respecto a inicio*	***HSPA+***	*% respecto a inicio*
Página web de inicio	3,741s		3,458s		2,408s		2,536s	
+Imágenes optimizadas	1,898s	49%	1,798s	48%	1,383s	43%	2,036s	20%
+CSS y JS Minificados	1,881s	50%	1,709s	51%	1,447s	40%	2,042s	19%
+Compresión Brotli	1,741s	53%	1,551s	55%	1,416s	41%	2,014s	21%

Como podemos ver con los resultados todas las optimizaciones aplicadas mejoran el tiempo de carga de la página web aunque en el caso de la conexión FTTH y HSPA+ se puede observar que las mediciones con las imágenes optimizadas es ligeramente mejor que las siguientes, esto es posible que se deba a las pequeñas fluctuaciones que se producen en internet con las horas del día y la saturación de algunos de los routers internos de la red.

Tabla 5–5 Speed Index tras las optimizaciones (Menos es mejor)

	ADSL	*% respecto a inicio*	***Media***	*% respecto a inicio*	***FTTH***	*% respecto a inicio*	***HSPA+***	*% respecto a inicio*
Página web de inicio	2660		2309		1507		2039	
+Imágenes optimizadas	1621	39%	1593	31%	1383	8,2%	1682	17,5%
+CSS y JS Minificados	1605	40%	1546	33%	1371	9,0%	1679	17,7%
+Compresión Brotli	1445	46%	1380	40%	1365	9,4%	1662	18,5%

Cuando medimos el Speed Index podemos comprobar que todas las optimizaciones usando cualquiera de las conexiones implican una mejora en el *Speed Index,* que nos intenta indicar como va a ser la percepción de velocidad de carga de nuestros visitantes. Por ello podemos estar seguros que ninguna de las optimizaciones está afectando de manera negativa a la página web.

En la Tabla 5–6 podemos comparar las diferentes mejoras en tamaño que hemos conseguido tras aplicar las optimizaciones y el porcentaje de ahorro respecto a la página sin optimizar.

Tabla 5–6 Tamaño total de la página web tras las optimizaciones

	ADSL	*% respecto a inicio*	***Media***	*% respecto a inicio*	***FTTH***	*% respecto a inicio*	***HSPA+***	*% respecto a inicio*
Página web de inicio	1109,7KB		1108,8KB		1078,34KB		1088,5KB	
+Imágenes optimizadas	408,87KB	63%	407,14KB	63%	405,62KB	62%	387,64KB	64%
+CSS y JS Minificados	363,84KB	67%	363,19KB	67%	363,84KB	66%	342,87KB	69%
+Compresión Brotli	271,35KB	76%	271,34KB	76%	271,34KB	66%	249,07KB	77%

Para finalizar hemos repetido el análisis de la página en GTmetrix tras realizar todos estos cambios, para ver la mejora desde la perspectiva del PageRank y de YSlow.

Figura 5-9 Análisis de la página usando GTmetrix después de las optimizaciones

Podemos comprobar en la figura 5-8 que hemos conseguido una puntuación casi perfecta en PageSpeed y una muy buena puntuación en YSlow.

6 CONCLUSIONES Y FUTURO DEL WPO

Mobile is important, and coming faster than most people in this room realize.

- Matt Cutts, 2014 -

Optimizar una página web, como se ha podido comprobar a lo largo de este trabajo, no es para nada complejo y está al alcance de todo el mundo. La instalación y configuración de software específico que nos permita aprovechar las ventajas de utilizar HTTP/2 o el nuevo algoritmo de compresión BROTLI pueden ser uno de los mayores problemas que un administrador se pueda encontrar, junto al correcto análisis acerca de que optimizaciones se deben aplicar en cada caso concreto.

Después de realizar este trabajo, hemos llegado a las conclusiones de que existen algunas optimizaciones muy sencillas de realizar como, por ejemplo, optimizar las imágenes y ajustarlas al tamaño con el que se utilizan. Este tipo de optimizaciones absolutamente triviales proporcionan una gran mejora en la experiencia del usuario, además de un gran ahorro en el ancho de banda del servidor.

También hemos podido verificar que es positivo minificar los recursos CSS y JavaScript, además de comprimirlos, preferiblemente, utilizando el algoritmo de compresión BROTLI.

Por otro lado, utilizar conexiones seguras HTTPS utilizando el protocolo HTTP/2 proporciona una navegación segura a nuestros visitantes y ofrece grandes mejoras en la velocidad de carga de la página, por ello es importante usarlo siempre que sea posible.

Destacar, que prácticamente todas las grandes innovaciones relacionadas con protocolos y algoritmos de compresión han sido realizadas por empleados de grandes empresas tecnológicas, y no por investigadores universitarios. Esto se debe a la gran inversión que realizan estas empresas en I+D relacionado con la optimización de páginas webs, debido a que una pequeña mejora puede suponer un ahorro de millones de dólares en ancho de banda, además de mejorar sus ventas o fidelizar a sus usuarios.

6.1 Líneas de futuro

El futuro de la optimización de páginas webs está garantizado, pues, una pequeña inversión puede suponer grandes beneficios. Por ello cada día más empresas de todos los tamaños se interesan por este campo.

Además, la cantidad de herramientas de monitorización de la velocidad y de herramientas para ayudar a realizar muchas de las optimizaciones está aumentando exponencialmente.

En los contenidos cubiertos por este trabajo nos hemos quedado sólo en la superficie de este apasionante mundo, centrándonos mayormente en las optimizaciones más básicas que pueden ser de mayor utilidad para las pequeñas y medianas empresas, pero las posibilidades de optimización son mucho más amplias, especialmente en páginas webs de gran complejidad.

Este trabajo podría ampliarse centrándose en algunas de las cosas que se nos han quedado en el tintero:

- La optimización de la forma en que se sirve el CSS separando la parte crítica, necesaria para la parte superior de la página web, del resto
- La creación de una *Progressive Web Apps* [46]
- El uso de *Service Workers* que nos permiten seguir ejecutando JavaScript en segundo plano, aunque se cierre la página web.
- Probar el uso de HTML5 WebSockets que sirven para establecer conexiones persistentes entre un navegador web y un servidor.

REFERENCIAS

[1] «The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search.,» [En línea]. Available: https://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/8523.

[2] «Shopzilla's Site Redo - You Get What You Measure,» [En línea]. Available: https://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/7709.

[3] «Improving Netflix Performance,» [En línea]. Available: https://conferences.oreilly.com/velocity/velocity2008/public/schedule/detail/3632.

[4] A. S. y M. C. , «Using Site Speed in Web Search Ranking,» [En línea]. Available: http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html.

[5] «Speed Index,» [En línea]. Available: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.

[6] «Web Page Test,» [En línea]. Available: https://www.webpagetest.org/.

[7] «GTMetrix,» [En línea]. Available: https://gtmetrix.com/.

[8] «Lighthouse,» [En línea]. Available: https://developers.google.com/web/tools/lighthouse/.

[9] Cisco, «Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper,» 28 Marzo 2017. [En línea]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[10] Mozilla Foundation, «CSS Media Queries,» [En línea]. Available: https://developer.mozilla.org/es/docs/CSS/Media_queries.

[11] w3schools, «Viewport,» [En línea]. Available: https://www.w3schools.com/css/css_rwd_viewport.asp.

[12] A. D. «Adding iOs test agents to a WebPageTest Instance,» [En línea]. Available: https://andydavies.me/blog/2013/03/05/adding-ios-test-agents-to-a-webpagetest-instance/.

[13] «New Relic Browser,» [En línea]. Available: https://newrelic.com/browser-monitoring.

[14] «Soasta mPulse,» [En línea]. Available: https://www.soasta.com/performance-monitoring/.

[15] «Pingdom Performance Monitoring,» [En línea]. Available: https://www.pingdom.com/product/performance-monitoring.

[16] «Google Analytics,» [En línea]. Available: https://analytics.google.com/.

[17] «Piwik,» [En línea]. Available: https://piwik.org/.

[18] S. Souders, Even Faster Web Sites: Performance Best Practices for Web Developers, O'reilly, 2009.

[19] «YSlow 2.0,» [En línea]. Available: http://yslow.es/.

[20] Google, «Reglas de PageSpeed Insights,» [En línea]. Available: https://developers.google.com/speed/docs/insights/rules?hl=es-419.

[21] The Chromium Project, «SPDY: An experimental protocol for a faster web,» [En línea]. Available: https://www.chromium.org/spdy/spdy-whitepaper.

[22] «Can I Use HTTP/2 protocol,» [En línea]. Available: https://caniuse.com/#feat=http2.

[23] M. B. R. P. y M. T. , «RFC7540: Hypertext Transfer Protocol Version 2 (HTTP/2),» [En línea]. Available: https://tools.ietf.org/html/rfc7540.

[24] M. B. R. P. y M. T. , «RFC7541: HPACK: Header Compression for HTTP/2,» [En línea]. Available: https://tools.ietf.org/html/rfc7541.

[25] Wikipedia, «CRIME,» [En línea]. Available: https://en.wikipedia.org/wiki/CRIME.

[26] I. G. «Innovating with HTTP 2.0 Server Push,» [En línea]. Available: https://www.igvita.com/2013/06/12/innovating-with-http-2.0-server-push/.

[27] W3C Working Draft, «HTML Resource Hints,» [En línea]. Available: https://www.w3.org/TR/resource-hints/#preconnect.

[28] «Can I Use Preconnect hint,» [En línea]. Available: https://caniuse.com/#search=preconnect.

[29] I. G. «Youtube: Is TLS Fast Yet?,» 2014. [En línea]. Available: https://www.youtube.com/watch?v=fQX2mJ11vCs.

[30] J. S. P. E. y H. T. , «RFC5057: Transport Layer Security (TLS) Session Resumption without Server-Side State,» IETF, [En línea]. Available: https://tools.ietf.org/html/rfc5077.

[31] Y. S. y H. T. , «RFC5723: Internet Key Exchange Protocol Version 2 (IKEv2),» IETF, [En línea]. Available: https://tools.ietf.org/html/rfc5723.

[32] A. L. N. M. y B. M. , «RFC7918: Transport Layer Security (TLS) False Start,» IETF, [En línea]. Available: https://tools.ietf.org/html/rfc7918.

[33] S. F. A. P. A. L. y E. S. , «RFC7301: Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension,» IETF, [En línea]. Available: https://tools.ietf.org/html/rfc7301.

[34] Wikipedia, «OCSP Stapling,» [En línea]. Available: https://en.wikipedia.org/wiki/OCSP_stapling.

[35] J. H. C. J. y A. B. , «RFC6797: HTTP Strict Transport Security (HSTS),» IETF, [En línea]. Available: https://tools.ietf.org/html/rfc6797.

[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee, «RFC2616: Hypertext Transfer Protocol -- HTTP/1.1,» Junio 1999. [En línea]. Available: https://www.ietf.org/rfc/rfc2616.txt.

[37] J. Alakuijala y Z. Szabadka, «RFC7932: Brotli Compressed Data Format,» Julio 2016. [En línea]. Available: https://tools.ietf.org/html/rfc7932.

[38] J. Alakuijala, E. Kliuchnikov, Z. Szabadka y L. Vandevenne, «Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM,» 09 2015. [En línea]. Available: https://www.gstatic.com/b/brotlidocs/brotli-2015-09-22.pdf.

[39] «Can I use Brotli,» [En línea]. Available: https://caniuse.com/#feat=brotli. [Último acceso: 23 05 2017].

[40] J. Daggett y M. C. Maxfield, «CSS Fonts Module Level 4,» [En línea]. Available: https://drafts.csswg.org/css-fonts-4/#descdef-font-face-font-display.

[41] S. Panicker, «Chrome 60 Beta: Paint Timing API, CSS font-display, and Credential Management API improvements,» 13 Junio 2017. [En línea]. Available: https://blog.chromium.org/2017/06/chrome-60-beta-paint-timing-api-css.html.

[42] Akamai, «Q1 2017 State of the internet Connectivity Report,» [En línea]. Available: https://www.akamai.com/es/es/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf.

[43] «Informe de seguimiento de los niveles de calidad de Servicio del Q1 2017,» [En línea]. Available: http://www.minetad.gob.es/telecomunicaciones/es-ES/Servicios/CalidadServicio/informes/Documents/Seguimiento_SAI_T1_17.pdf.

[44] SamKnows, «Quality of Broadband in Europe,» [En línea]. Available: http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=4996.

[45] «Lossy Image Optimization,» [En línea]. Available: https://kraken.io/docs/lossy-optimization.

[46] Google Developers, «Progressive Web Apps,» [En línea]. Available: https://developers.google.com/web/progressive-web-apps/.

[47] «Emular y probar otros navegadores,» [En línea]. Available: https://developers.google.com/web/tools/chrome-devtools/device-mode/testing-other-browsers.

ÍNDICE DE CONCEPTOS

ANEXOS

Anexo A: Instalación de una instancia privada de WebPageTest en Amazon AWS

Para hacerlo lo más fácil posible vamos a utilizar Amazon EC2 para lanzar una instancia sobre Linux que tendrá la aplicación que hace las peticiones y recopila los datos, el maestro, y otra instancia sobre Windows que será la que realice los tests en el navegador web, el esclavo.

Antes de empezar es necesario disponer de una cuenta de Amazon con información de pago asociada, ya que lanzar instancias no es gratuito.

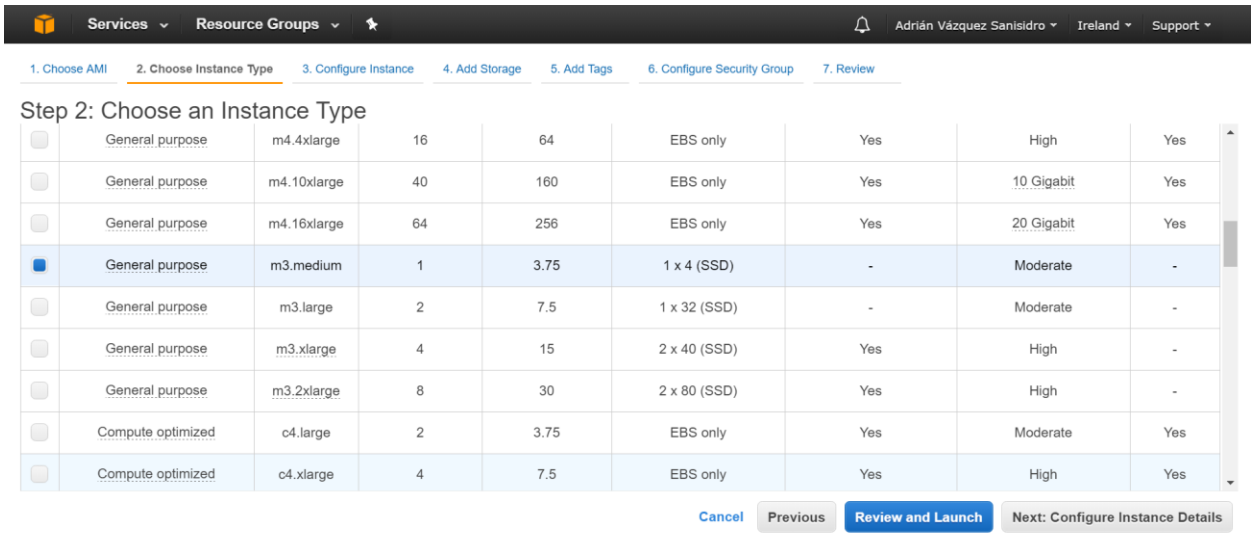
Para facilitar la instalación vamos a hacer uso de los *Amazon Machine Images*, también llamados AMI, que son imagenes de las particiones del disco duro de una instancia una vez que tiene instalado el sistema operativo y todos los programas necesarios para realizar una función en concreto. Gracias a la existencia de los AMI podemos ahorrar muchas horas de instalación y configuración.

Paso 1: Configurar el maestro

Actualmente existen instancias del maestro de *WebPageTest* preconfiguradas para cada región donde Amazon AWS tiene servidores, por lo que empezamos seleccionando la que se encuentra en Europa:

- eu-west-1 (Europa/Irlanda): ami-9978f6ee [1]

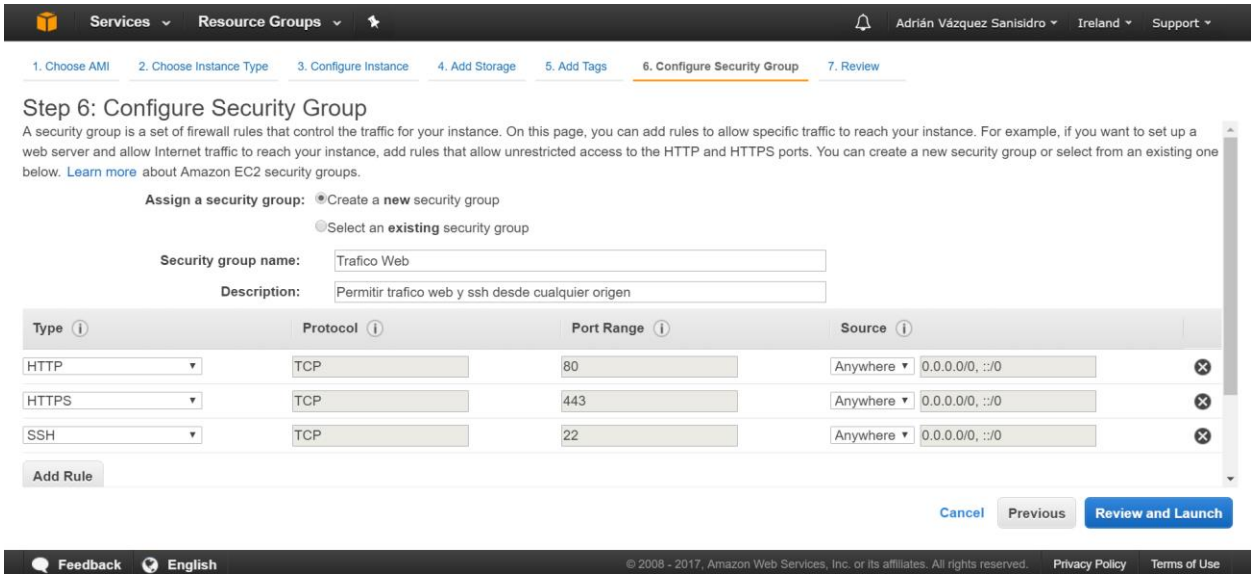
El proceso de creación es bastante sencillo, pasamos a elegir el tipo de instancia que queremos. En principio vamos a coger un m3.medium que tiene una capacidad de procesamiento elevada.



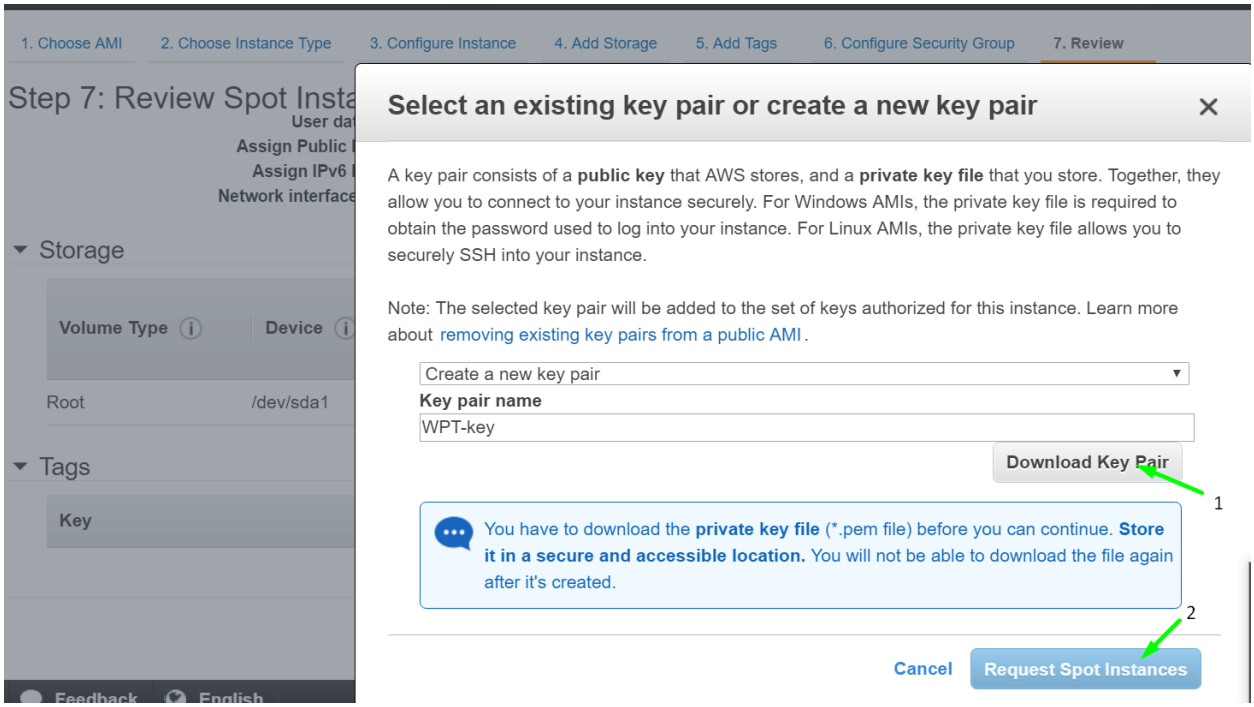
	General purpose	m4.4xlarge	16	64	EBS only	Yes	High	Yes
	General purpose	m4.10xlarge	40	160	EBS only	Yes	10 Gigabit	Yes
	General purpose	m4.16xlarge	64	256	EBS only	Yes	20 Gigabit	Yes
■	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate	-
	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate	-
	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High	-
	General purpose	m3.2xlarge	8	30	2 x 80 (SSD)	Yes	High	-
	Compute optimized	c4.large	2	3.75	EBS only	Yes	Moderate	Yes
	Compute optimized	c4.xlarge	4	7.5	EBS only	Yes	High	Yes

Pulsamos en *Next: Configure Instance Details* y dejamos todo como está por defecto hasta el Paso 6 (*Configure Security Group*) donde debemos configurar que permitimos el tráfico HTTP y HTTPS, así como el tráfico SSH, en el ejemplo lo configuramos como desde cualquier origen pero es aconsejable, por cuestiones de seguridad, permitir sólo tráfico desde el rango IP donde se vaya a utilizar.

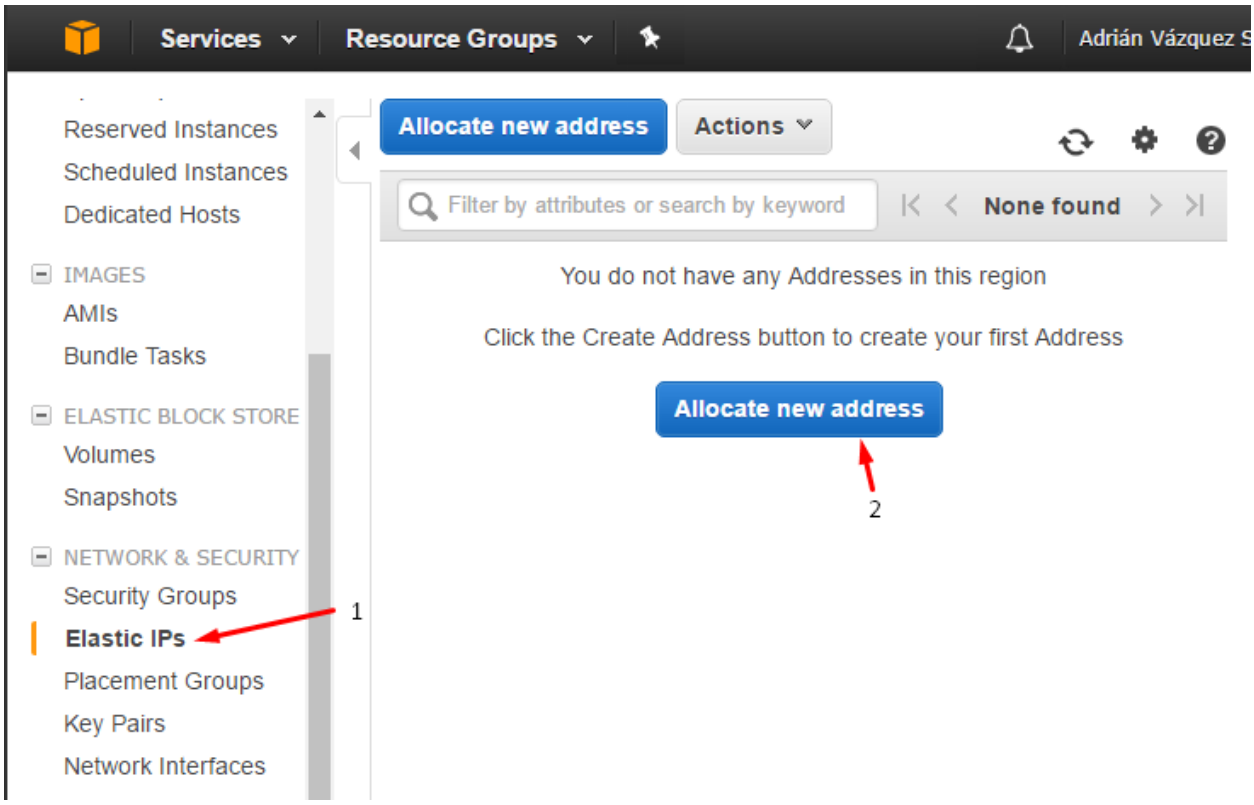
[1] https://console.aws.amazon.com/ec2/v2/home?region=eu-west-1#LaunchInstanceWizard:ami=ami-9978f6ee



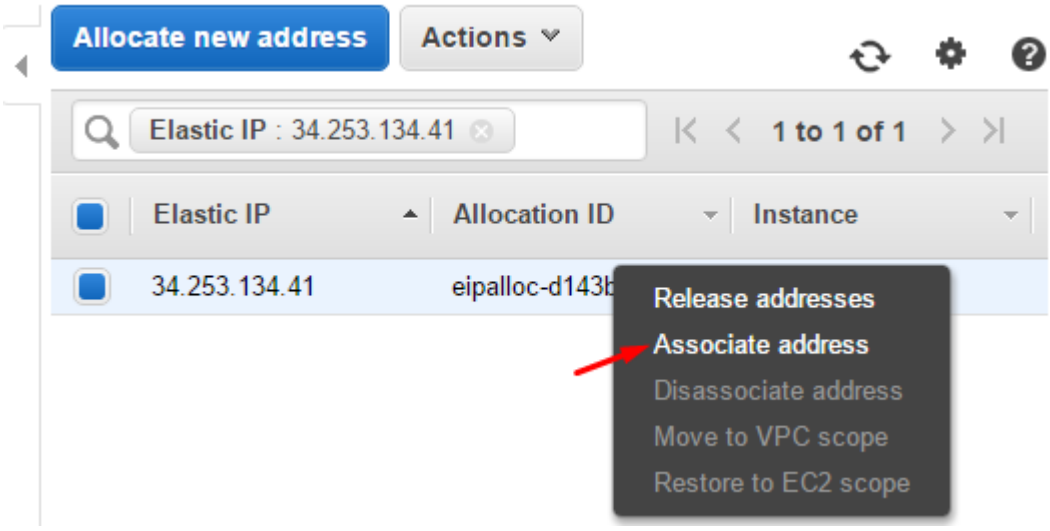
Una vez configurado, creamos un par de claves nuevas y nos las descargamos para poder acceder a nuestro maestro más adelante. Seguidamente creamos la instancia.



Ahora necesitamos que nuestra instancia maestro tenga una dirección IP fija para que las instancias esclavo sepan donde comunicarse, esto se hace asignandole una *Elastic IP*. Para ello en la sección de *NETWORKING & SECURITY* nos vamos a Elastic IPs y creamos una dirección nueva.



Tras ello asociamos la IP recién creado a nuestra instancia maestro:



Paso 2: Configurar los esclavos

Para ello tenemos que empezar buscando los AMI de las instancias que más nos interesen dependiendo de la region donde hayamos creado nuestra instancia maestro[2].

Para simplificar listamos a continuación algunas de las instancias más comunes:

[2] Lo podemos buscar en https://github.com/WPO-Foundation/webpagetest/blob/master/www/settings/locations.ini.EC2-sample buscando el código AMI del navegador que nos interese

- Como nuestro maestro está en eu-west-1 (Europa/Irlanda)
  - Chrome, Firefox, Safari e IE11: ami-a3a81dd0[3]
  - IE8: ami-00b18074[4]
  - IE7: ami-70b18004[5]

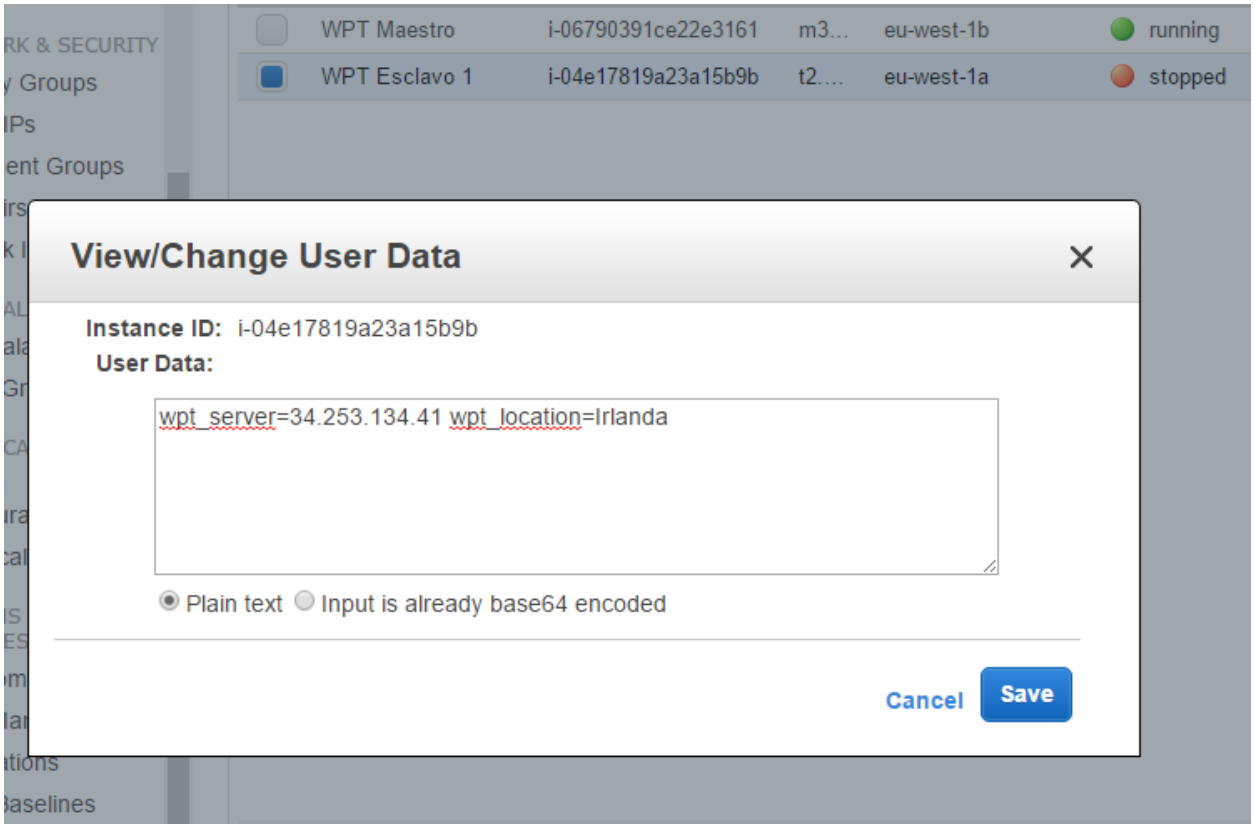
Creamos la instancia de la misma manera que antes, sólo que a la hora de indicar la política de seguridad abrimos el puerto 3389 (RDP) en vez del 22 (SSH) ya que las instancias esclavo se ejecutan sobre Windows.

Paso 3: Emparejamos la instancia esclavo a la instancia maestro

Para esto lo primero que tenemos que hacer es parar nuestra instancia esclavo que acabamos de crear. Después pulsamos con el botón derecho en la instancia esclavo y dentro del menu *Instance Settings* pulsamos en *View/Change User Data*.

Introducimos la siguiente línea, sustituyendo la ip del maestro por la ip elástica que hemos conseguido al final del paso 1, y le ponemos un nombre al esclavo para poder distinguirlo posteriormente:

wpt_server=IP_DEL_MAESTRO wpt_location=NOMBRE_DEL_ESCLAVO



[3] https://console.aws.amazon.com/ec2/v2/home?region=eu-west-1#LaunchInstanceWizard:ami=ami-a3a81dd0
[4] https://console.aws.amazon.com/ec2/v2/home?region=eu-west-1#LaunchInstanceWizard:ami=ami-00b18074
[5] https://console.aws.amazon.com/ec2/v2/home?region=eu-west-1#LaunchInstanceWizard:ami=ami-70b18004

Paso 4: Configuramos la instancia maestro

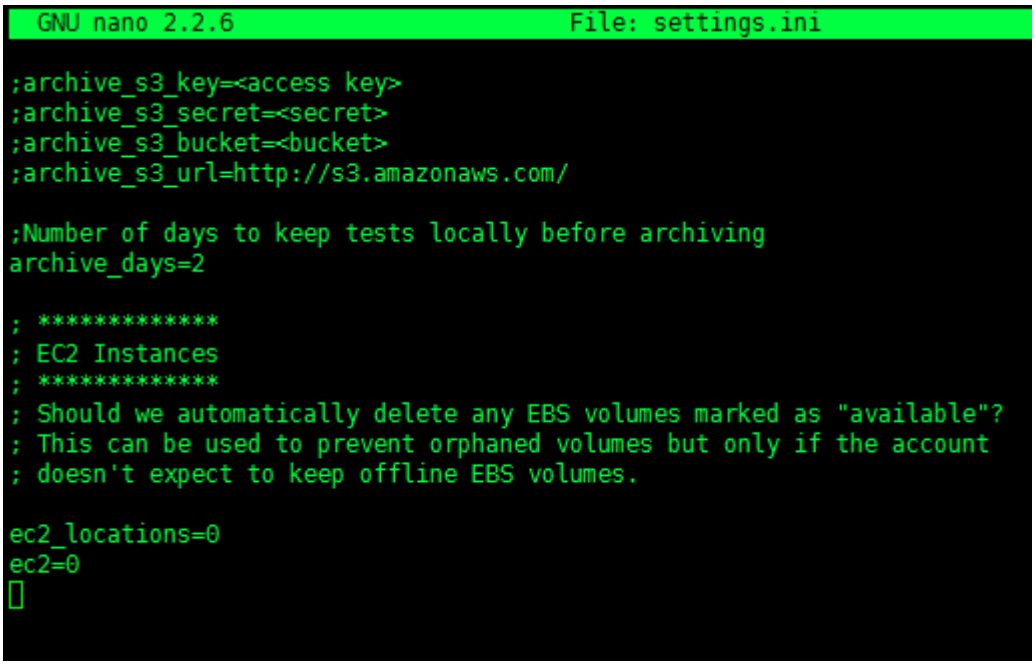
Para ello nos tenemos que conectar usando un cliente SSH a la IP elástica de nuestra instancia maestro.

En el caso de usar el famoso cliente SSH *Putty*, necesitamos transformar la clave SSH que hemos descargado en el paso 1 usando *PuttyGen*. Cuando nos pida el usuario que queremos utilizar, introducimos *ubuntu*.

Tenemos que editar 2 archivos .ini que se encuentran en el siguiente directorio:

/var/www/webpagetest/www/settings

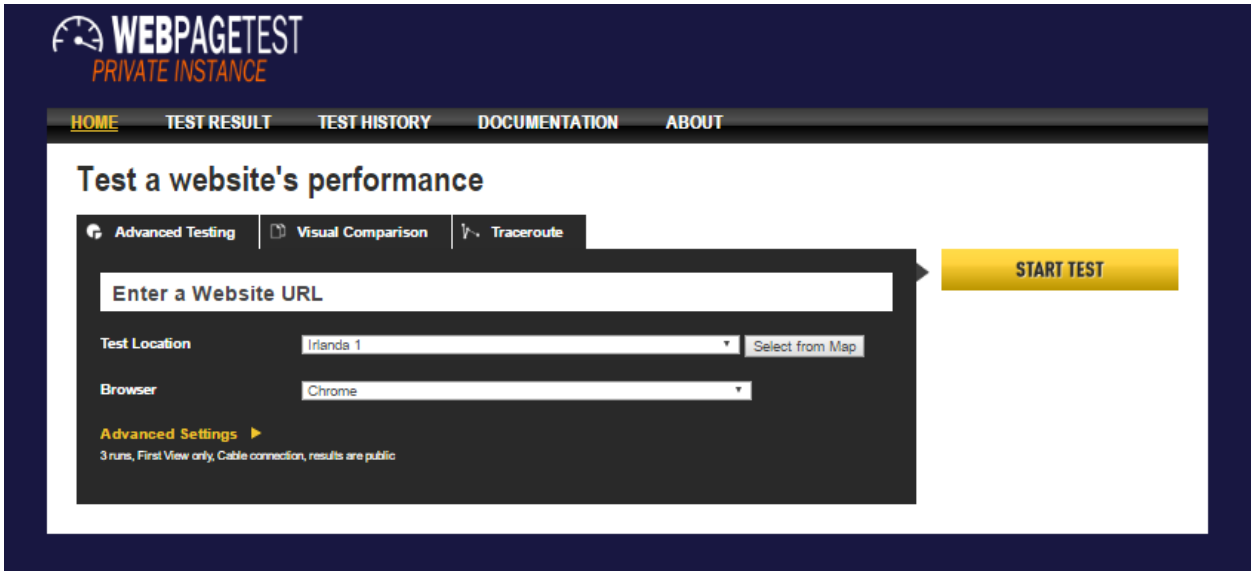
Primero abrimos el fichero **settings.ini**, y al final del mismo ponemos ec2_locations y ec2 a 0.



Después abrimos **locations.ini**, y añadimos las siguientes líneas cambiando donde pone NOMBRE_DEL_ESCLAVO por el nombre que le asignamos en el paso 3.

```
[locations]
1=EC2
default=EC2
[EC2]
1=NOMBRE_DEL_ESCLAVO-WPT_wptdriver
label="Irlanda"
default= NOMBRE_DEL_ESCLAVO-WPT_wptdriver
[NOMBRE_DEL_ESCLAVO-WPT_wptdriver]
browser=Chrome,Firefox, IE11
label="Instancia con chrome, firefox e IE11"
```

Despues de hacer estos cambios nos dirigimos a http://IP_ELÁSTICA_DEL_MAESTRO/install y veremos que todo está correcto y ya podemos utilizar nuestra instancia privada de WebPageTest dirigiendonos a http://IP_ELÁSTICA_DEL_MAESTRO


WEBPAGETEST
PRIVATE INSTANCE
HOME
TEST RESULT
TEST HISTORY
DOCUMENTATION
ABOUT
Test a website's performance
Advanced Testing
Visual Comparison
Traceroute
Enter a Website URL
START TEST
Test Location
Irlanda 1
Select from Map
Browser
Chrome
Advanced Settings
3 runs, First View only, Cable connection, results are public

Anexo B: Script para obtener los resultados de la API de WPT

Para realizar este script vamos a utilizar NodeJS debido a que es el lenguaje de programación en el que existe la librería, también llamado módulo en NodeJS, más completa para poder acceder a la API de nuestra instancia privada de WebPageTest que hemos instalado en el anexo anterior

Instalar node.js en CentOS 6

Para ello vamos a utilizar el repositorio de nodesource, el cual añadimos con el siguiente comando:

```
curl --silent --location https://rpm.nodesource.com/setup_4.x | bash -
```

Tras ello lo instalamos, usando root, con:

```
yum -y install nodejs
```

Instalar el modulo de la API de WebPageTest

Esto es muy sencillo de realizar debido al funcionamiento de NodeJS, sólo tenemos que ejecutar el siguiente comando en la carpeta donde vayamos a ubicar nuestro script que haga uso de este módulo:

```
npm install webpagetest -g
```

Script escrito en node.js

```
1. //Incluimos la librería de la API de WPT
2. var WebPagetest = require('webpagetest');
3. //Incluimos la librería para escribir en archivos
4. var fs = require('fs');
5.
6. var datosEscribir
7.
8. console.log('---- Empezando nuevo test: '+process.argv[2]+' ----')
9.
10. //Función que se encarga de formatear los resultados y escribirlos en
  el CSV además de mostrar por la salida estandar los datos más relevantes
11. function resultados(err, result, conexion) {
12.
13.    //Mostramos por consola los resultados de la primera carga (First
  view) (Se usaría `repeatView` para la segunda carga)
14.    //Los resultados que vamos a usar son las medias y aunque los
  números iniciales tienen decimales al ser una media puede tenerlos, por
  eso las redondeamos.
15.   console.log(err || 'Load
  time:', Math.round(result.data.average.firstView.loadTime)+'s')
16.   console.log('First
  byte:', Math.round(result.data.average.firstView.TTFB)+'s')
17.   console.log('Start
  render:', Math.round(result.data.average.firstView.render)+'s')
18.   console.log('Speed
  Index:', Math.round(result.data.average.firstView.SpeedIndex))
19.   console.log('DOM
  elements:', Math.round(result.data.average.firstView.domElements),'elemen
  tos')
20.
```

```
21.   console.log('(Doc complete)
  Requests:', Math.round(result.data.average.firstView.requestsDoc), 'petic
  iones')
22.   console.log('(Doc complete) Bytes
  in:', Math.round(result.data.average.firstView.bytesInDoc), 'bytes')
23.
24.   console.log('(Fully loaded)
  Time:', Math.round(result.data.average.firstView.fullyLoaded)+'s')
25.   console.log('(Fully loaded)
  Requests:', Math.round(result.data.average.firstView.requestsFull), 'peti
  ciones')
26.   console.log('(Fully loaded) Bytes
  in:', Math.round(result.data.average.firstView.bytesIn), 'bytes')
27.
28.   console.log('Waterfall
  view:', result.data.runs[1].firstView.images.waterfall)
29.
30. //Formateamos los datos que vamos a guardar según nuestra plantilla
31.   datosEscribir = process.argv[2]+';'+Math.round(result.data.average.fi
  rstView.TTFB) +'; '+ Math.round(result.data.average.firstView.render) +';
  '+ Math.round(result.data.average.firstView.loadTime) +';
  '+ Math.round(result.data.average.firstView.fullyLoaded) +';;
  '+  Math.round(result.data.average.firstView.SpeedIndex) +';;
  '+ Math.round(result.data.average.firstView.requestsDoc) +' ;
  '+ Math.round(result.data.average.firstView.bytesInDoc) +';;
  '+ Math.round(result.data.average.firstView.requestsFull) +' ;
  '+ Math.round(result.data.average.firstView.bytesIn) +';;;;
  '+ Math.round(result.data.average.repeatView.TTFB) +';
  '+ Math.round(result.data.average.repeatView.render) +';
  '+ Math.round(result.data.average.repeatView.loadTime) +';
  '+ Math.round(result.data.average.repeatView.fullyLoaded) +';;
  '+ Math.round(result.data.average.repeatView.SpeedIndex) +';;
  '+ Math.round(result.data.average.repeatView.requestsDoc) +' ;
  '+ Math.round(result.data.average.repeatView.bytesInDoc) +';;
  '+ Math.round(result.data.average.repeatView.requestsFull) +' ;
  '+ Math.round(result.data.average.repeatView.bytesIn) +'\r\n'
32.         //Escribimos la nueva línea en el archivo CSV
33.         fs.appendFile('resultados-
  '+conexion+'.csv', datosEscribir, 'utf8', function (err) {
34.                   if (err) {
35.                     console.log('Ha ocurrido algún error - El fichero
  no se ha guardado o se ha corrompido.');
36.                   } else{
37.                     console.log('Se ha añadido esta medición al CSV.');
38.                   }
39.                 });
40.  };
41.
42. //Nos conectamos a nuestra instancia de WPT indicando su ip y su API
  Key
43. var wpt = new WebPagetest('34.253.134.41', '33f6b472561edfcf6130b2a65b6
  87104f9ed5d62');
44.
45. //Ejecutamos el test adsl y comprobamos con pollResults cada 10
  segundos si el resultado ya esta disponible
46. console.log('Realizando 20 tests con velocidades ADSL');
47. wpt.runTest('https://apocalipsisz.es/', {runs:20, pollResults: 10, vide
  o: 0, connectivity:'custom', location:'irlanda_wptdriver', bwdown:'8500',
   bwup:'700', latency:'60'}, function (err, result) {
48.                 console.log('-- RESULTADOS velocidades ADSL --');
49.                 resultados(err, result, 'adsl');
50.         });
```

```
51.
52.
53. console.log('Realizando 20 tests con velocidades medias');
54. //Ejecutamos el test con velocidad media y comprobamos con pollResults
    cada 10 segundos si el resultado ya esta disponible
55. wpt.runTest('https://apocalipsisz.es/', {runs:20, pollResults: 10, vide
    o: 0, connectivity:'custom', location:'irlanda_wptdriver', bwdown:'15500'
    , bwup:'1500', latency:'45'}, function (err, result) {
56.                 console.log('-- RESULTADOS velocidades medias --');
57.                 resultados(err, result, 'media');
58.         });
59.
60.
61. console.log('Realizando 20 tests con velocidades FTTH');
62. //Ejecutamos el test con velocidad ftth y comprobamos con pollResults
    cada 10 segundos si el resultado ya esta disponible
63. wpt.runTest('https://apocalipsisz.es/', {runs:20, pollResults: 10, vide
    o: 0, connectivity:'custom', location:'irlanda_wptdriver', bwdown:'5000',
     bwup:'5000', latency:'20'}, function (err, result) {
64.                 console.log('-- RESULTADOS velocidades FTTH --');
65.                 resultados(err, result, 'ftth');
66.         });
67.
68.
69. console.log('Realizando 20 tests con velocidades HSPA+');
70. //Ejecutamos el test movil y comprobamos con pollResults cada 10
    segundos si el resultado ya esta disponible
71. wpt.runTest('https://apocalipsisz.es/', {runs:20, pollResults: 10, vide
    o: 0, connectivity:'custom', location:'irlanda_wptdriver', bwdown:'10000'
    , bwup:'3000', latency:'100', mobile: 1}, function (err, result) {
72.                 console.log('-- RESULTADOS velocidades HSPA+ --');
73.                 resultados(err, result, 'movil');
74.         });
```

Explicación y ejecución del script

Este script realiza 20 tests de velocidad para cada una de las cuatro conexiones que hemos modelado en la sección 5.1 usando la API de nuestra instancia de WebPageTest que hemos instalado en el Anexo A: Instalación de una instancia privada de WebPageTest en Amazon AWS.

Muestra por pantalla y guarda en 4 archivos CSV diferentes (uno por tipo de connexion) una nueva línea con la media aritmética de los tiempos obtenidos en los 20 tests.

El script acepta un parámetro que sería el nombre de la prueba y que escribiremos en la primera celda de la nueva línea del CSV para poder saber posteriormente que estabamos probando en cada medida.

Utilizarlo es muy sencillo, solo tenenos que guardar el archivo con la extension .js, presuponemos que se llamará prueba.js, en la misma carpeta donde instalamos el modulo de la API de WebPageTest, y usar el siguiente comando:

node prueba.js Descripcion-De-La-Prueba

Archivo con los resultados de las pruebas

El script detallado anteriormente nos guarda los resultados en 4 archivos CSV, que se pueden abrir con cualquier programa de hojas de cálculo o incluso importar los datos a una base de datos. Cada uno de los 4 archivos corresponde a una de las conexiones que hemos modelado en el apartado 5.1.

A continuación, se muestra una captura de parte del archivo con los resultados de las pruebas realizadas:

Tipo de prueba	First View								
	Times					Doc Complete		Fully Loaded	
	First Byte	Start Render	Load Time	Fully Loaded	Speed Index	Requests	Bytes in	Request	Bytes in
Base-Apache2.4-PHP5.4-No-Cache	403	1173	2366	2442	1622	28	1128938	29	1139602
Base-Apache2.4-PHP5.5-No-Cache	403	1197	2406	2480	1644	28	1129015	29	1139679
Base-Apache2.4-PHP5.6-No-Cache	403	1198	2389	2465	1631	28	1128978	29	1139642
Base-Apache2.4-PHP7.0-No-Cache	356	1118	2332	2408	1507	26	1104218	27	1114882
Base-Apache2.4-PHP7.1-No-Cache	345	1133	2334	2407	1589	28	1129614	29	1140278
Base-Apache2.4-PHP5.4-HTTPS-Nc	471	1177	2520	2590	1882	26	1162129	27	1172613
Base-Apache2.4-PHP5.5-HTTPS-Nc	471	1190	2529	2602	1919	26	1162050	27	1172541
Base-Apache2.4-PHP5.6-HTTPS-Nc	463	1180	2579	2650	1964	26	1162156	27	1172644
Base-Apache2.4-PHP7.0-HTTPS-Nc	400	1115	2437	2508	1913	26	1161977	27	1172465
Base-Apache2.4-PHP7.1-HTTPS-Nc	409	1114	2433	2505	1983	26	1163876	27	1174360
Imagenes-lossless-mismo-tamaño	409	1097	2295	2367	1864	26	1084794	27	1095532
Imagenes-lossy-mismo-tamaño	405	1073	1558	1633	1459	26	637987	27	648727
Imagenes-lossy-tamaño-exacto	406	1065	1261	1334	1329	26	445715	27	456451
CSS-JS-minificados	408	1062	2376	2448	2039	25	1114148	26	1124889
Compresion-Deflate	434	1066	2348	2418	2135	26	1010570	27	1021311
Compresion-Brotli	423	1076	2330	2400	1949	26	1007529	27	1018267
Css-y-JS-ordenados	427	1207	2671	2739	2226	25	1099762	25	1110502

Anexo C: Instalación y uso de BROTLI en un servidor Linux con Apache

Actualmente hay múltiples implementaciones del algoritmo de compresión BROTLI, pero la mayoría de ellas tienen poco soporte y dan algunos problemas. Por lo que hemos optado por una precompresión usando la implementación original de los empleados de Google, para ello la compilaremos desde el código fuente que es Open-Source y podemos encontrarlo en GitHub

```
git clone https://github.com/google/brotli
cd ./brotli
./configure && make
```

Uso de la herramienta Brotli

Desde la carpeta donde hemos descargado y compilado la herramienta ejecutamos el siguiente comando modificando "/path/to/file" por la ruta al archivo que queramos comprimir y la opción *-q* nos permite elegir el nivel de compresión (de 1 a 11):

```
./bin/brotli -q 7 /path/to/file
```

De esta forma obtendremos el archivo comprimido con el mismo nombre y en la misma ubicación que el original, pero con el sufijo ".br"

Debemos repetir esta operación por cada archivo que queramos comprimir (HTML, CSS o JavaScript), si estamos en una web donde estos recursos se actualizan con cierta frecuencia sería buena idea crear un Script Bash que nos automatice el proceso de recomprimir todos los recursos.

Cómo gestionar en Apache los archivos comprimidos con BROTLI

Vamos a utilizar el mod_rewrite de Apache para enviar los archivos comprimidos con BROTLI sólo a los usuarios de navegadores compatibles y al resto les enviaremos el archivo comprimido con DEFLATE.

Para ello tenemos que editar el archivo .htaccess que se encuentra en el directorio raíz de la página web y añadir las siguientes líneas:

```
# Comprobamos que este instalado el mod_rewrite y lo inicializamos
<IfModule mod_rewrite.c>
  RewriteEngine on
    # BROTLI
      # Comprobamos si el navegador nos ha dicho que acepta BROTLI
        RewriteCond %{HTTP:Accept-encoding} br
        # y que la petición es de un recurso que puede ser comprimido
        RewriteCond %{REQUEST_URI} .*\.(css|js)
        # trás ello comprobamos que exista el recurso comprimido con BROTLI
        RewriteCond %{REQUEST_FILENAME}.br -s
        # si existe reescribimos lo que vamos a mandar a dicha versión
        RewriteRule ^(.+) $1.br
        # Indicamos el tipo mime correcto para que el navegador sepa que tipo
de archivo es y nos aseguramos que Apache no intente recomprimir los archivos
        RewriteRule "\.css\.br$" "-" [T=text/css,E=no-brotli,E=no-gzip]
        RewriteRule "\.js\.br$" "-" [T=application/javascript,E=no-
```

```
brotli,E=no-gzip]


        <FilesMatch "\.(css|js)\.br$">

            # Nos aseguramos que el modulo mime envie una cabecera de que el
idioma es Portugues (Brasileño) debido a que el archivo acaba en .br

            RemoveLanguage .br

            # Configuramos la cabecera que indica que el contenido está
comprimido con BROTLI

            Header set Content-Encoding br

            # Obligamos a los servidores Proxies a mantener la versión
comprimida con BROTLI separada de la normal

            Header append Vary Accept-Encoding

        </FilesMatch>

</IfModule>

    #DEFLATE

      #Comprobamos que este instalado el mod_deflate

      <IfModule mod_deflate.c>

        # Comprimimos los recursos HTML, CSS, JavaScript, XML y las fuentes
no comprimidas anteriormente

        AddOutputFilterByType DEFLATE application/javascript

        AddOutputFilterByType DEFLATE application/rss+xml

        AddOutputFilterByType DEFLATE application/vnd.ms-fontobject

        AddOutputFilterByType DEFLATE application/x-font

        AddOutputFilterByType DEFLATE application/x-font-opentype

        AddOutputFilterByType DEFLATE application/x-font-otf

        AddOutputFilterByType DEFLATE application/x-font-truetype

        AddOutputFilterByType DEFLATE application/x-font-ttf

        AddOutputFilterByType DEFLATE application/x-javascript

        AddOutputFilterByType DEFLATE application/xhtml+xml

        AddOutputFilterByType DEFLATE application/xml

        AddOutputFilterByType DEFLATE font/opentype

        AddOutputFilterByType DEFLATE font/otf

        AddOutputFilterByType DEFLATE font/ttf

        AddOutputFilterByType DEFLATE image/svg+xml

        AddOutputFilterByType DEFLATE image/x-icon

        AddOutputFilterByType DEFLATE text/css

        AddOutputFilterByType DEFLATE text/html

        AddOutputFilterByType DEFLATE text/javascript

        AddOutputFilterByType DEFLATE text/plain

        AddOutputFilterByType DEFLATE text/xml

       </IfModule>
```

Anexo D: Instalación de HTTP/2 en Apache

Para poder usar HTTP/2 en Apache es necesario utilizar al menos la versión 2.4.25 de Apache. La mayoría de las distribuciones de Linux actuales tienen en sus repositorios una versión más antigua y es necesario actualizarla.

En un servidor con CentOS, se puede instalar esta versión compilando desde el código fuente, tal como se explica en muchos sitios[6], pero esto implica, posteriormente, problemas para mantener el servidor seguro con las futuras actualizaciones. Además, si utilizamos CentOS 6 también tenemos que compilar manualmente la última versión de OpenSSL para que sea compatible con ALPN, que es uno de los requisitos impuestos por los navegadores web para utilizar HTTP/2.

Algunos paneles de control para servidores web, como Cpanel o Plesk tienen sus propios repositorios que facilitan la instalación y mantenimiento de los programas más utilizados. El modulo mod_http2, que nos da soporte de HTTP/2 en Apache, ha sido marcado como estable a partir de la versión 2.4.26 de Apache que ha sido lanzada en junio de 2017, por ello se espera que a lo largo del año las distintas distribuciones de Linux empiecen a actualizar su versión de Apache a una compatible con HTTP/2.

En Cpanel, por ejemplo, es muy sencillo de instalar, ya que el instalador que han creado se encarga de actualizar Apache a la versión adecuada e incluye una versión moderna de OpenSSL, aunque se espera que lo simplifiquen aún más a corto plazo, pudiendo instalarlo directamente desde el panel de control. Actualmente tenemos que ejecutar los siguientes comandos:

```
yum install ea4-experimental
yum install ea-apache24-mod_http2
```

En Plesk, otro de los paneles más utilizados, HTTP/2 está soportado a partir de Plesk 12.5 utilizando nginx en vez de Apache. Aunque para tener soporte ALPN es necesario que estemos utilizando CentOS 7, ya que la versión de OpenSSL que incluye CentOS 6 no es compatible. Para activar el soporte de HTTP/2 tan sólo hay que usar el siguiente comando:

```
/usr/local/psa/bin/http2_pref enable
```

Una vez que tengamos una versión de Apache superior a la 2.4.25, para activar el módulo mod_http2, tenemos que ejecutar los siguientes comandos en la terminal:

```
a2enmod http2
apachectl restart
```

Para añadir soporte HTTP/2 a una de las webs que tengamos alojadas en el servidor, tenemos que asegurarnos que está configurada correctamente para utilizar HTTPS, y agregar en el archivo de configuración de apache la palabra h2 a la línea de protocolos:

```
Protocols h2 http/1.1
```

Por lo que el archivo de configuración de cada dominio quedaría de la siguiente manera:

```
<VirtualHost *:443>
  Protocols h2 http/1.1
  ServerAdmin hola@nuestrodominio.com
  ServerName nuestrodominio.com
  ...
</VirtualHost>
```

[6] https://www.tunetheweb.com/performance/http2/

Configuración de TLS en Apache 2.4

Por otro lado, es aconsejable que configuremos adecuadamente los cifrados utilizados, añadiendo las siguientes líneas al archivo de configuración de Apache:

```
  SSLProtocol ALL -SSLv2 -SSLv3
  SSLHonorCipherOrder On
  SSLCipherSuite ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES12
8:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS
```

Fuera del bloque VirtualHost ponemos la línea:

```
  SSLStaplingCache shmcb:/tmp/stapling_cache(128000)
```

Y dentro del bloque VirtualHost debemos poner las siguientes líneas, para activar SSL Stapling, configurar un timeout de 5 segundos al hacer las peticiones a los servidores OCSP y para no guardar las respuestas no validas:

```
  SSLUseStapling on
  SSLStaplingResponderTimeout 5
  SSLStaplingReturnResponderErrors off
```

Quedando de la siguiente manera:

```
  SSLStaplingCache shmcb:/tmp/stapling_cache(128000)
  <VirtualHost *:443>
    Protocols h2 http/1.1
    ServerAdmin hola@nuestrodominio.com
    ServerName nuestrodominio.com

    SSLUseStapling on
    SSLStaplingResponderTimeout 5
    SSLStaplingReturnResponderErrors off
    ...
  </VirtualHost>
```

Y tras realizar todos estos cambios, reiniciamos Apache:

```
  apachectl restart
```

La versión 2.4.25 de Apache, que necesitamos para HTTP/2, incluye por defecto el soporte de los dos métodos para utilizar *Session resumption* y el soporte de *TLS False Start*, por ello no hemos indicado instrucciones adicionales.