14-6-2017

PROYECTO INTEGRADO – HTTP/2

PROY. FINAL

JUAN LUIS RAMIREZ VAQUERO
2º ASIR

ÍNDICE

1.- INTRODUCCION

- El mundo de La informática siempre está en continua mejora e innovación y ya tenemos nuevo protocolo “HTTP/2”. Como podemos observar es un protocolo bastante nuevo y llega con la intención de actualizar el protocolo “HTTP/1.1”, con el cual es completamente compatible.
- Primero podemos observar que “HTTP/2” no modifica la semántica de la aplicación de http, en general mantiene todos los conceptos básicos, tales como los métodos, HTTP, códigos de estados, URI y campos de cabecera.
- Sin embargo, “HTTP 2.0” introduce innumerables mejoras, como el uso de una única conexión, la compresión de cabeceras o el servicio “server push”.
- Inicialmente surgió el protocolo SPDY para implementar HTTP. El objetivo de dicho protocolo a nivel de sección era reducir la latencia. Dicho protocolo logro mejorar casi un 60% la velocidad de carga y hasta un 55% las conexiones protegidas con certificados SSL.

1.1.- **Línea temporal HTTP**

1.2.- **Características del protocolo.**

- Este protocolo llega con el claro objetivo de mejorar las carencias existentes en las anteriores versiones.
- Sus características principales son las siguientes:

1.2.1.- <u>Una única conexión</u>

- Con HTTP/1.x para cargar cualquier contenido web es necesario el uso de múltiples conexiones TCP simultáneas para poder descargar todos los elementos de dicha web. En cambio, HTTP 2.0 utiliza una única conexión para ofrecer múltiples solicitudes y respuestas en paralelo. Teniendo en cuenta que cada página web puede contener objetos **HTML, CSS, JavaScript, imágenes, vídeo, etc…** la diferencia de trabajo entre utilizar una única conexión o utilizar varias es elevada.

1.2.2.- <u>Eliminación de información redundante</u>

- Otro cambio de gran relevancia en HTTP 2.0 es la eliminación de información redundante cuyo objetivo es evitar el envío de datos repetidos durante una misma conexión, así conseguiremos que se consuman menos recursos, obteniendo una menor latencia.

1.2.3.- Multiplexación

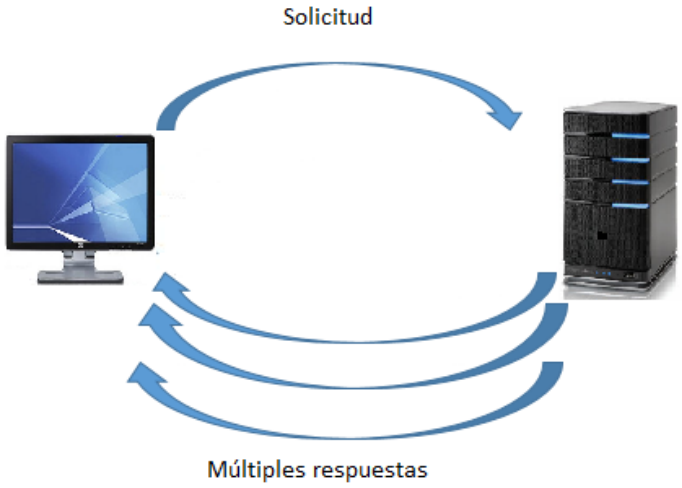
- Con HTTP/1.1 el navegador envía una petición y debe esperar la respuesta del servidor para poder enviar la siguiente solicitud. Aquí viene el problema, las webs modernas suelen tener más de 100 objetos, por lo que el retardo es grande. La solución que introduce HTTP 2.0 a este problema es la denominada Multiplexación.
- ¿Qué es la multiplexación?, La multiplexación permite enviar y recibir varios mensajes al mismo tiempo optimizando la comunicación. Con la multiplexación se consigue reducir el número de conexiones mejorando considerablemente la velocidad de carga y disminuyendo la carga de los servidores web.

1.2.4.- Se trata de un protocolo binario

- La ventaja que tiene el uso de un protocolo binario es la facilidad para encontrar el comienzo y el final de cada "frame", que es algo realmente complicado en cualquier protocolo de texto. Además, los protocolos binarios son mucho más simples y por lo tanto son menos propensos a tener errores que los protocolos de texto utilizados por las versiones anteriores a HTTP 2.0.

1.2.5.- Servicio "server push"

- El servicio "server push" también conocido como "cache push", se basa en estimaciones para que el servidor sea capaz de enviar información al usuario antes de que éste la solicite para que la información esté disponible de forma inmediata.
- La forma de actuar del servidor es enviar varias respuestas a una única solicitud del cliente, es decir, además de la respuesta a la solicitud original, el servidor puede enviar recursos adicionales. Esto es así porque una página web está formada por una gran cantidad de archivos referenciados que gracias al servicio "server push" el servidor envía tras recibir una única solicitud ahorrando mensajes innecesarios.



- HTTP 2.0 contiene un campo denominado 'Ajustes' con el que el cliente puede indicar si desea o no obtener los recursos que proporciona el servicio 'server push'.

1.2.6.- Compresión de cabeceras para transmitir menos información

- Con las versiones anteriores a HTTP 2.0, las cabeceras de los mensajes de solicitud eran de texto claro, sin ningún tipo de compresión. El problema aparece como consecuencia del incremento de tamaño que sufren estas cabeceras por los **user-agent** de los navegadores, al uso de cookies (también deben aparecer en los mensajes de solicitud), etc. Además, cuando HTTP/1.1 envía una petición, debe esperar la respuesta del servidor para poder enviar la siguiente solicitud, aumentado mucho el retardo sufrido.

1.2.6.1.- ¿Qué es user-agent?

- Un agente de usuario es una aplicación informática que funciona como cliente en un protocolo de red; el nombre se aplica generalmente para referirse a aquellas aplicaciones que acceden a la World Wide Web (WWW). Los agentes de usuario que se conectan a la Web pueden ser desde navegadores web hasta los "web crawler" de los buscadores y teléfonos móviles.
- Asimismo, hay que tener en cuenta que cuando un cliente pide numerosas peticiones a un mismo servidor, los encabezamientos apenas cambian, por lo que se envía mucha información redundante, es decir, se podría omitir.
- Con HTTP 2.0 las cabeceras experimentan compresiones, con lo que se obtienen mejores tiempos de respuesta y también se mejora la eficiencia (sobre todo en terminales móviles). El algoritmo empleado para realizar la compresión de cabeceras es ***HPACK***.
    - ¿Qué es ***HPACK***?, es un algoritmo simple y poco flexible que se basa en eliminar campos de cabecera redundantes, además de prevenir posibles vulnerabilidades.

1.2.7.- Priorización de flujos

- Un mensaje HTTP se puede dividir en múltiples fragmentos en su recorrido desde el cliente hasta el servidor o desde servidor al cliente. El orden y el retardo con el que estas tramas llegan a su destino son fundamentales, dado que algunos objetos de las webs son más importantes que otros. Nos interesará que los objetos más relevantes cuenten con algún tipo de prioridad.
- Para poder 'controlar' la prioridad que tienen las tramas, HTTP 2.0 permite asignar a cada flujo un peso (entre 1 y 256) y una dependencia. Debemos ser conscientes de que las prioridades pueden variar durante la ejecución.

1.2.8.- No requiere cifrado TLS.

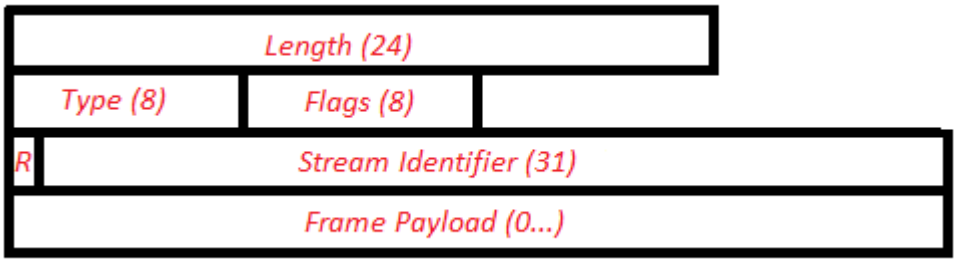
- En HTTP/2 el uso de cifrado TLS (**Transport Layer Security**) es opcional. De todos modos, un gran número de fabricantes de software (Firefox, Internet Explorer o Google Crome) ya han anunciado que sus implementaciones solo soportarán HTTP 2.0 sobre TLS.

1.2.8.1.- ¿Qué es TLS?

- Es un protocolo criptográfico de la capa de transporte (de criptografía asimétrica), que proporciona comunicaciones seguras por la red. El uso de TLS añade un retardo adicional.

1.3.- **Formato de la trama**

- Las frames pueden tener múltiples tamaños, hasta un límite máximo de 16 kb, a no ser que el cliente y el servidor lleguen a un acuerdo para utilizar un tamaño de frame mayor, en cuyo caso el máximo tamaño al que puede llegar es de 16 MB. Pese a esta posibilidad las frames no suelen superar los 16 kb de tamaño.
- Todas las frames comienzan con una cabecera de 9 octetos fijos seguidos de la carga útil de longitud variable.
- El formato es el siguiente:



  - Length: 24 bits que indican la longitud de dicho frame sin contar los 9 bytes de la cabecera.
  - Type: 8 bits que nos informan de cómo se interpreta el resto de la trama. A pesar de que hoy día hay pocos tipos de frames disponibles, los 8 bits dejan espacio para 256 formatos diferentes de frames.
  - Flags: 8 bits que determinan el contenido de los flags.
  - R: se trata de un bit reservado. Debe permanecer con valor ‘0’ al ser enviado y debe ser ignorado cuando se recibe.
  - Stream Identifier: Se trata de un identificador de flujo de 32 bits. El bit más significativo siempre es 0.
  - Payload: contiene los datos, es de longitud variable.

1.4.- **Definiciones de frame.**

1.4.1.- Cabeceras.

- Las cabeceras se utilizan para iniciar un flujo, y pueden contener los siguientes campos:
    - Pad Length (8 bits): contiene la longitud de la trama. Se mide en octetos.
    - E (1bit): Bandera de un bit. Depende de la prioridad.
    - Stream Dependency (31 bits): se trata de un identificador de flujo.
    - Weight (8 bits): representa el nivel de prioridad, su valor puede variar de 1 a 256.
    - Header Block Fragment: es un fragmento del bloque de cabecera.
    - Padding: octetos para padding (tiene un tamaño variable).
    - Quedan definidos los siguientes indicadores:
    - END_STREAM (0x1): El último mensaje en una secuencia utiliza este campo para indicar el final de la transferencia de datos.
    - END_HEADERS (0x4): Campo que indica el final de las cabeceras. En este caso la trama contiene un bloque de cabecera completo.
    - PADDED (0x8): Cuando se establece, indica que el campo de Longitud Pad y cualquier tipo de relleno están presentes.
    - PRIORITY (0x20): Con este campo se puede indicar al servidor la prioridad que tiene el mensaje. Podrá ser modificada durante la ejecución.

1.4.2.- Prioridad

- Las frames utilizan campos de prioridad (type = 0x2) para comparar distintos tipos de tramas dando preferencias a unas respecto de otras.
- Esta prioridad puede ser modificada en cualquier momento, sea cual sea el estado de la frame, pero es aconsejable que durante el intercambio de cabeceras no sea modificado su valor.
- En el caso en el que, por cualquier motivo, se reciba una prioridad sin contenido se produce el siguiente error: “Protocol error”.
- En el caso en el que se reciban en primer lugar los contenidos, y en segundo lugar sus prioridades, estás no se tendrán en cuenta a la hora de procesar las tramas.

1.5.- **Stream**

- Las stream son secuencias de frames independientes y bidireccionales que se intercambian entre cliente y servidor durante una conexión HTTP 2.0. Una única conexión HTTP/2 puede contener múltiples streams activos de forma simultánea que serán procesados en el destino en el orden en el que fueron enviados.
- Los puntos finales no se coordinan en la creación de flujos, pero cabe destacar que los streams pueden ser cerrados por cualquiera de los puntos finales de la comunicación.

- Todos los stream contienen un identificador (un número entero) asignado por el origen y parten de un estado 'inactivo'.
- Pueden ocurrir los siguientes sucesos:
    - Si se envían o reciben cabeceras, el stream pasa al estado 'abierto' y, en algunos casos, puede pasar a un estado 'medio-cerrado'.
    - Si se produce un envío de 'push_promise' el stream pasa a un estado 'reservado (local)'.
    - Cuando se recibe el envío 'push_promise' se pasa a un estado 'reservado (a distancia)'.

1.6.- **Códigos de errores.**

- Los códigos de error son campos de 32 bits utilizados para notificar los distintos fallos que pueden producirse. Los códigos son los siguientes:
    - NO_ERROR (0x0): La condición asociada no es el resultado de un error. Por ejemplo, podría usarse para indicar que el ordenador ha cerrado la conexión de forma inesperada.
    - PROTOCOL_ERROR (0x1): Error producido en el protocolo específico.
    - INTERNAL_ERROR(0x2): El punto final encontró un error interno inesperado.
    - FLOW_CONTROL_ERROR (0x3): Se da cuando el punto final detecta que su par incumple el protocolo de control de flujo.
    - SETTINGS_TIMEOUT (0x4): Tiene lugar cuando el punto final envía ajustes de marco, pero no recibe una respuesta a tiempo.
    - STREAM_CLOSED (0x5): Cierra la secuencia actual y libera todos los recursos.
    - FRAME_SIZE_ERROR (0x6): El punto final recibe una trama con un tamaño no válido.
    - REFUSED_STREAM (0x7): El servidor no procesa la respuesta.
    - CANCEL (0x8): Campo utilizado por el punto final para indicar que la conexión ya no es necesaria. Se termina el stream.
    - COMPRESSION_ERROR (0x9): Error al realizar la compresión de cabeceras. La conexión es cerrada por un motivo poco usual.
    - CONNECT_ERROR (0xa): La conexión establecida se ha cerrado de forma inesperada.
    - ENHANCE_YOUR_CALM (0xb): El punto final detecta que su par está exhibiendo un comportamiento que podría generar una carga excesiva.
    - INADEQUATE_SECURITY (0xc): Se activa cuando no se cumplen los requisitos mínimos de seguridad.
    - HTTP_1_1_REQUIRED (0xd): El punto final requiere que se utilice HTTP versión 1.1 en lugar de usarse HTTP versión 2.0.

1.7.- **Intercambio de mensajes con HTTP 2.0.**

- HTTP 2.0 está destinada a ser lo más compatible posible con los usos actuales de HTTP. Esto significa que, desde la perspectiva de aplicación, las características del protocolo apenas sufren cambios.
- Cada mensaje de solicitud o respuesta de HTTP 2.0 consta de:
    - Una o varias cabeceras que deben ser enviadas al comienzo de la comunicación, es lo primero que en ser enviado. Son tramas de información.
    - En segundo lugar, debe enviarse el contenido del mensaje (payload).
    - Por último, se envían los posibles campos de cola (por ejemplo, el campo padding).

1.8.- **Seguridad**

1.8.1.- Padding como mecanismo de seguridad

- En HTTP/2 el padding puede ser utilizado para ocultar el tamaño exacto del contenido de la trama, con la idea de ocultar la compresión de los datos más sensibles.
- Debe tomarse en consideración que el uso del padding de forma redundante puede llegar a ser contraproducente, además de que su uso, en el mejor de los casos, sólo hace que sea más complicado para un atacante hacerse con la información, pero en ningún caso consigue hacerlo inevitable.

1.8.1.1.- ¿Qué es el padding?

- Establece la anchura de algunas o todas las zonas de relleno de los elementos.
- La propiedad padding es una de las "propiedades shorthand" que define CSS y que se utilizan para establecer de forma abreviada el valor de una o más propiedades individuales.

1.8.2.- Vulnerable a un ataque de denegación del servicio

- En HTTP 2.0 se permiten múltiples intercambios de stream por cada conexión TCP. El problema que puede darse es de denegación del servicio (sobrecarga de los recursos computacionales del sistema), debido a que se permite un total de $2^{\wedge 31}$ intercambios de stream por conexión.
- Un atacante que quisiera sobrecargar la red podría abrir múltiples conexiones en las que se intercambiasen numerosos stream. Por este motivo se debe limitar el intercambio de stream por conexión.

1.8.3.- Vulnerabilidad por el uso del servicio 'server push'

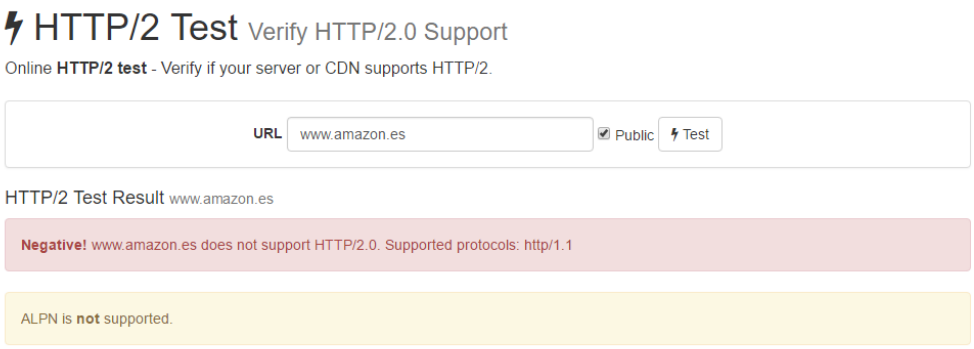
- Uno de los posibles problemas que nos podemos encontrar en HTTP 2.0 es debido al uso del servicio ‘server push’.
- Un atacante podría aprovechar las múltiples respuestas que realiza el servidor a una única solicitud, el uso de memorias caché… con el fin de poder acceder a contenidos web que fueron de uso público y ahora son de uso privado (que requieren de privilegios para poder ser accedidos).

1.9.- **Herramienta para comprobar protocolo HTTP**

- Podemos comprobar si una página se encuentra implementado con http/2, vamos a utilizar la herramienta "https://tools.keycdn.com/http2-test", vamos a poner dos ejemplos uno el cual esta implementado en "HTTP/2" y otro en "HTTP/1.1".

1.9.1.- **HTTP/1.1**

- Una página que es de imaginar por la gran carga que tiene, el número de visitas y que lo que se puede pedir de una página como esta es rapidez y se encuentre en el protocolo más rápido, no es otra más que "www.amazon.es":



1.9.2.- **HTTP/2**

- Y podemos comprobar que una página la cual era de esperar utilizara este protocolo como es "www.google.es", si utiliza este protocolo.

1.9.2.1.- ¿Qué es ALPN?

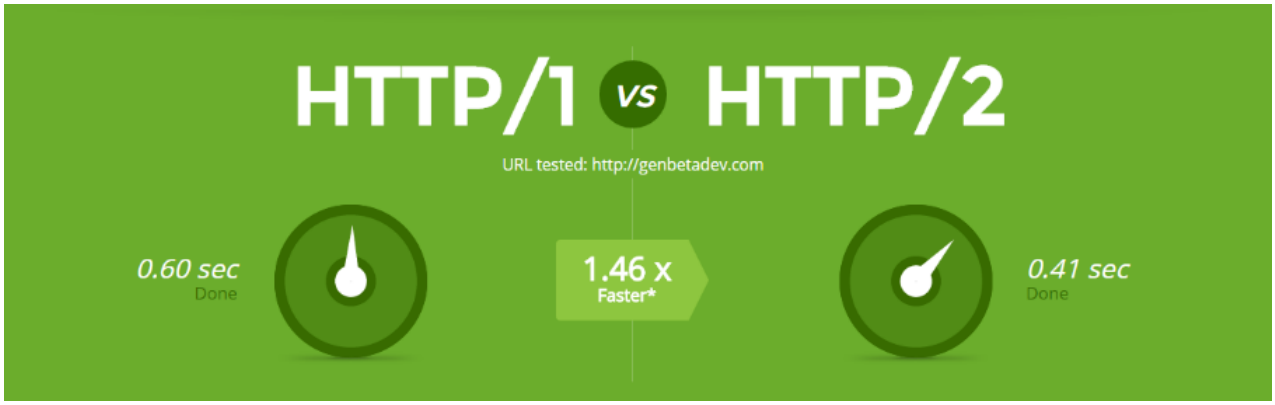
- ALPN sirve para que navegador y servidor decidan qué protocolo usar… en una conexión segura.
- ALPN cierra el puzzle de la imposición de Google quien hace unos años trató de crear una alternativa a http: el "SPDY". Luego, cuando HTTP2 cobró fuerza y se aceptó como estándar, Google tiró para atrás y dijo que Chrome ya no iba a soportar "SPDY" sino que sólo iba a funcionar con ALPN.
- En lo que se traduce es que muchos de los ISP van a tener que correr para instalar actualizaciones que permitan ALPN para poder usar HTTP2.
- Por ejemplo, CentOS 6 y CentOS 7, que son sistemas operativos muy comunes en servidores web, vienen con una versión anterior que no soporta ALPN así que, o tenemos conocimientos técnicos, o hasta dentro de un año no vamos a tener soporte fácil de HTTP2 en este tipo de servidores.

1.10.- **Implementación del protocolo en la actualidad**

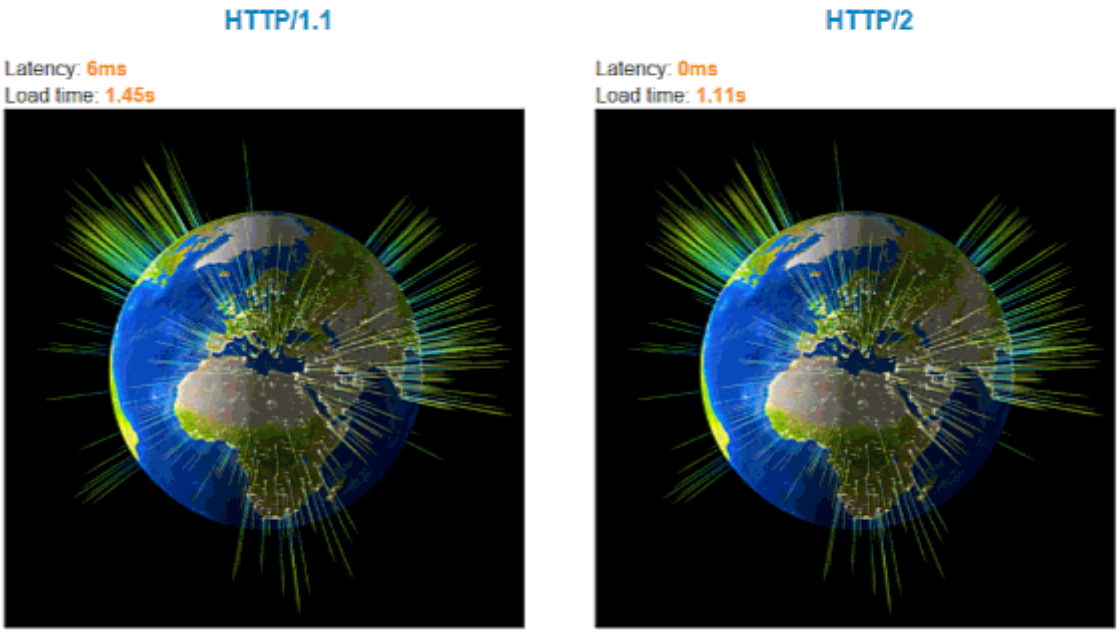
- A pesar de que el borrador del protocolo se proporcionó a la IETF el 11 de febrero de 2015, algunos navegadores ya lo han implementado, al igual que algunos servidores. Por poner un ejemplo, Google comenzó en abril de 2014 a ofrecer servicios HTTP 2.0 en unos pocos servidores a modo de prueba.
- En la actualidad, las últimas versiones de los navegadores más actualizados (Firefox v36, Chrome v40 y Explorer v11) ya soportan este nuevo protocolo. En Google Chrome y en Firefox se ha decidido que el protocolo HTTP 2.0 sólo sea utilizado en conexiones criptografiadas.
- En países como Reino Unido (53,1%), Alemania (58,02%) o Canadá (50,35%) las peticiones con HTTP 2.0 ya superan a las peticiones que usan los protocolos anteriores. Estos datos demuestran cómo HTTP 2.0 se empieza a afianzar pese a ser un protocolo tan reciente.

2.- DIFERENCIAS ENTRE HTTP/1 Y HTTP/2.0

- Como ya hemos explicado, HTTP/2.0 es un protocolo el cual permite características tales como:
    - multiplexed streams
    - server push
    - la compresión de HEADERS
    - formato binario.
- Como podemos ver en la siguiente foto, se trata de un protocolo bastante más rápido que el predecesor. Según este estudio el protocolo “http/2” es casi 2 puntos más rápido que “http/1”.
- También podemos observar que mientras “http/1” tarda 0.60 segundos en cargar la página, si la página está en “http/2” tarda 0.41 segundos, es decir, 0.20 menos que el protocolo “http/1”



- En la siguiente foto podemos ver otro ejemplo de la diferencia entre los dos protocolos, en este caso se observar la diferencia de latencia en ambos protocolos.



2.1.- **¿Qué es la latencia?**

- En redes informáticas de datos, se denomina latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.
- Otros factores que influyen en la latencia de una red son:
    - El tamaño de los paquetes transmitidos.
    - El tamaño de los búferes dentro de los equipos de conectividad.

3.- INSTALACION Y CONFIGURACION SERVIDOR APACHE CON HTTP/2

- Ahora vamos a configurar dos máquinas para las pruebas que más adelante realizaremos, y vamos a instalar una maquina con un servidor apache con “http/2” y otra máquina con “http/1.1” (el cual viene habilitado por defecto).
- Para poder implantar http/2 en un servidor apache, primero tenemos que tener un certificado ssl instalado, para ello vamos a realizar un certificado auto firmado con "OpenSSL".

3.1.- **Generar certificado auto firmado.**

- Un certificado SSL es un certificado digital utilizado por el protocolo para el encriptamiento de la información.
- Este certificado es proporcionado por un proveedor autorizado (Verisign, Thawte, Comodo, etc…) y es enviado al cliente por el servidor con quien estamos estableciendo una conexión segura.
- Hay muchos servicios que utilizan este protocolo, algunos ejemplos pueden ser: HTTPS, SMTPS, IMAPS, SSH, POP3S, etc…

3.2.- **Instalar OpenSSL**

```
root@http2:/home/usuario# apt install openssl
```

3.3.- **Creamos una clave privada**

- La llave privada nos será útil para la generación del certificado. Una vez creado, nuestro certificado SSL dependerá de esta llave para la implementación del mismo en cualquier servicio que requiera una conexión segura.
- Por ejemplo, vamos a crear una clave de 1024 bits:

```
root@http2:/home/usuario# openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
....++++++
.....................................................................++++++
e is 65537 (0x010001)
```

3.4.- **Crear CSR**

- Un CSR es la base para un certificado SSL, en él se definen datos como el dominio, organización, ubicación, información de contacto, entre otros.
- Es importante destacar que estos pasos también son necesarios cuando vas a adquirir un certificado SSL de un proveedor autorizado, durante la gestión del mismo, el proveedor va a solicitar este archivo para crear tu certificado. Por lo tanto, debemos tener mucho cuidado en que la información que ingresamos sea correcta.

- Para generar el CSR debemos ejecutar el siguiente comando, si te fijas uno de los parámetros que introducimos es la clave privada.

```
root@http2:/home/usuario# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]: Seville
Locality Name (eg, city) []: Seville
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Juanlu Ramirez
Organizational Unit Name (eg, section) []: Juanlu Ramirez
Common Name (e.g. server FQDN or YOUR name) []: Juanlu
Email Address []: admin@juanluramirez.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3.5.- **Generamos certificado SSL**

- Para generar el certificado SSL vamos a necesitar tanto la llave privada como el CSR que acabamos de crear.
- Para generar el certificado SSL debemos ejecutar el siguiente comando:

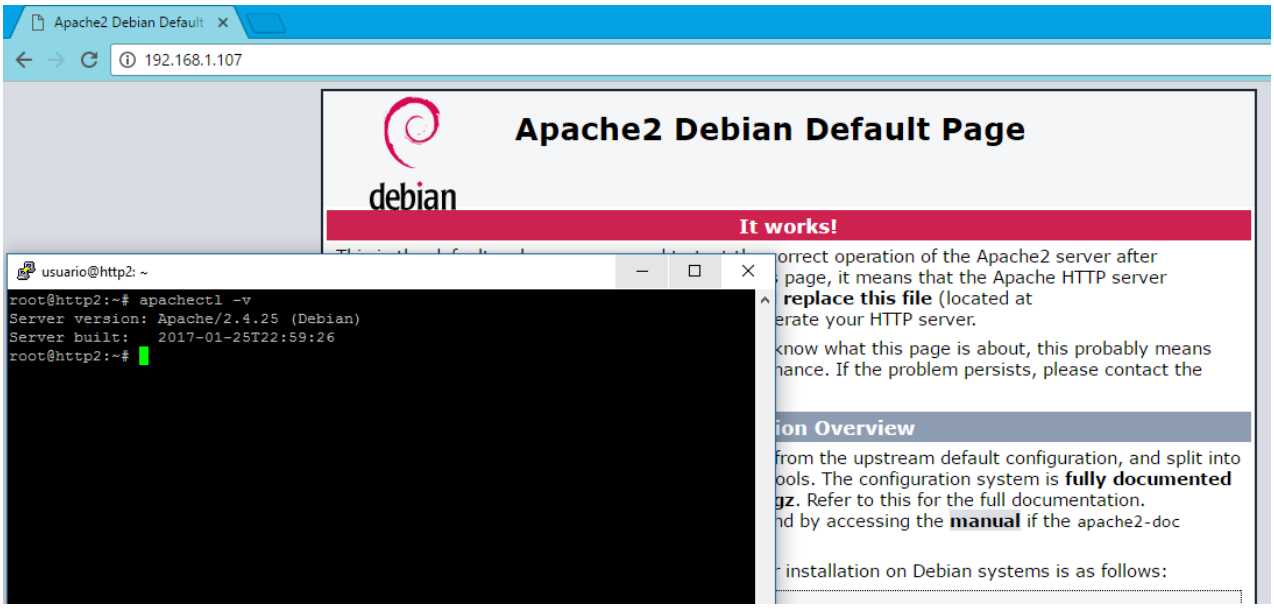
```
root@http2:/home/usuario# openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=C = ES, ST = Seville, L = Seville, O = Juanlu Ramirez, OU = Juanlu Ramirez, CN = Juanlu, emailAddress = admin@juanluramirez.com
Getting Private key
```

3.6.- **Instalación pila LAMP http/2.**

- Al realizar en el último punto del proyecto una instalación de un "CMS" como va a ser "Wordpress" vamos a instalar, tanto php como MariaDB.

```
root@http2: ~# apt install apache2 php mariadb-server
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
El paquete indicado a continuación se instaló de forma automática y ya no es necesario.
  linux-image-3.16.0-4-amd64
Utilice «apt autoremove» para eliminarlo.
Se instalarán los siguientes paquetes adicionales:
  apache2-bin apache2-data apache2-utils galera-3 gawk libaio1
  libapache2-mod-php7.0 libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libjemalloc1 liblua5.2-0 libmariadbclient18 libmpfr4 libreadline5
  libterm-readkey-perl mariadb-client-10.1 mariadb-client-core-10.1
  mariadb-common mariadb-server-10.1 mariadb-server-core-10.1 mysql-common
  php-common php7.0 php7.0-cli php7.0-common php7.0-json php7.0-opcache
  php7.0-readline rsync socat
Paquetes sugeridos:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom gawk-doc
  php-pear libclone-perl libmldbm-perl libnet-daemon-perl
  libsql-statement-perl libipc-sharedcache-perl mariadb-test netcat-openbsd
  tinyca
Se instalarán los siguientes paquetes NUEVOS:
  apache2 apache2-bin apache2-data apache2-utils galera-3 gawk libaio1
  libapache2-mod-php7.0 libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libjemalloc1 liblua5.2-0 libmariadbclient18 libmpfr4 libreadline5
  libterm-readkey-perl mariadb-client-10.1 mariadb-client-core-10.1
  mariadb-common mariadb-server mariadb-server-10.1 mariadb-server-core-10.1
  mysql-common php php-common php7.0 php7.0-cli php7.0-common php7.0-json
  php7.0-opcache php7.0-readline rsync socat
0 actualizados, 38 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 23,0 MB de archivos.
Se utilizarán 175 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
```

- Ya tendremos el servidor apache instalado y operativo.



- Copiamos los certificados a "/etc/ssl/certs"

```
root@http2:/home/usuario# cp server.crt /etc/ssl/certs/
root@http2:/home/usuario# cp server.key /etc/ssl/certs
```

- Y habilitamos el modulo SSL en apache:

```
root@http2:/home/usuario# a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.
To activate the new configuration, you need to run:
  systemctl restart apache2
```

- Y modificamos el vhosts por defecto de SSL "/etc/apache2/sites-available/default-ssl", borramos todo el contenido y añadimos lo siguiente.

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
    ServerName www.http2.com
    ServerAdmin admin@juanluramirez.com
    DocumentRoot /var/www/html
    <Directory />
       Options FollowSymLinks
       AllowOverride None
    </Directory>
    <Directory /var/www/html>
```

```
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>
  ErrorLog ${APACHE_LOG_DIR}/error.log
  LogLevel warn
  CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined
  SSLEngine on
  SSLCertificateKeyFile /etc/ssl/certs/server.key
  SSLCertificateFile /etc/ssl/certs/server.crt
  #SSLCACertificateFile /etc/ssl/certs/bundle.crt
  BrowserMatch "MSIE [2-6]" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0
  # MSIE 7 and newer should be able to use keepalive
  BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
 </VirtualHost>
</IfModule>
```

- Habilitamos el vhosts "a2ensite default-ssl" y reiniciamos el servidor apache "systemctl restart apache2".

3.7.- **Habilitar http2**

- Primero instalamos el paquete nghttp2

```
root@http2:/home/usuario# apt install nghttp2
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libc-ares2 libev4 libjansson4 libspdylay7 nghttp2-client nghttp2-proxy nghttp2-server
Se instalarán los siguientes paquetes NUEVOS:
  libc-ares2 libev4 libjansson4 libspdylay7 nghttp2 nghttp2-client nghttp2-proxy nghttp2-server
0 actualizados, 8 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 793 kB de archivos.
Se utilizarán 2.035 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
```

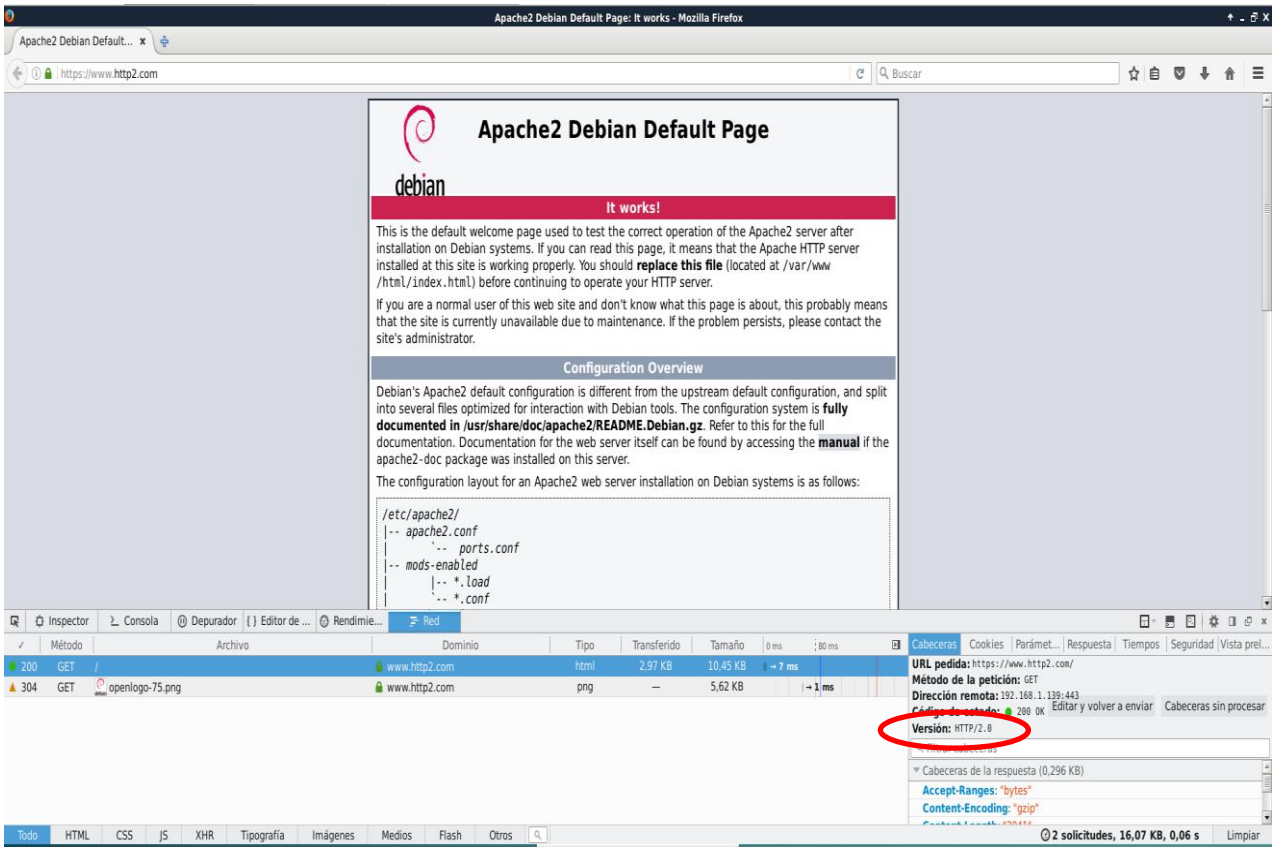
- Habilitamos el módulo http2

```
root@http2:/home/usuario# a2enmod http2
Enabling module http2.
To activate the new configuration, you need to run:
  systemctl restart apache2
```

- Y añadimos la siguiente linea a nuestro fichero "default-ssl", justo detras de los certificados especificados.

```
Protocols h2 http/1.1
```

- Y ya solo queda reiniciar el servidor web "systemctl restart apache2"



4.- ESTUDIO DE RENDIMIENTO ENTRE HTTP/2 Y HTTP/1.1, CON GRÁFICAS.

- En el apartado anterior hicimos la instalación y configuración del servidor web apache con “http/2”, ahora vamos a realizar la instalación de la pila “LAMP” en el servidor web donde vamos a utilizar “http/1” para poder hacer una comprobación correcta.

4.1.- **Instalación LAMP**

- Instalamos la pila LAMP en la maquina “http1”.

```
root@http1:/home/usuario# apt install apache2 mariadb-server php
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
El paquete indicado a continuación se instaló de forma automática y ya no es necesario.
  linux-image-3.16.0-4-amd64
Utilice «apt autoremove» para eliminarlo.
Se instalarán los siguientes paquetes adicionales:
  apache2-bin apache2-data apache2-utils galera-3 gawk libaio1
  libapache2-mod-php7.0 libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libjemalloc1 liblua5.2-0 libmariadbclient18 libmpfr4 libreadline5
  libterm-readkey-perl mariadb-client-10.1 mariadb-client-core-10.1
  mariadb-common mariadb-server-10.1 mariadb-server-core-10.1 mysql-common
  php-common php7.0 php7.0-cli php7.0-common php7.0-json php7.0-opcache
  php7.0-readline rsync socat
Paquetes sugeridos:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom gawk-doc
  php-pear libclone-perl libmldbm-perl libnet-daemon-perl
  libsql-statement-perl libipc-sharedcache-perl mariadb-test netcat-openbsd
  tinyca
Se instalarán los siguientes paquetes NUEVOS:
  apache2 apache2-bin apache2-data apache2-utils galera-3 gawk libaio1
  libapache2-mod-php7.0 libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libjemalloc1 liblua5.2-0 libmariadbclient18 libmpfr4 libreadline5
  libterm-readkey-perl mariadb-client-10.1 mariadb-client-core-10.1
  mariadb-common mariadb-server mariadb-server-10.1 mariadb-server-core-10.1
  mysql-common php php-common php7.0 php7.0-cli php7.0-common php7.0-json
  php7.0-opcache php7.0-readline rsync socat
0 actualizados, 38 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 23,0 MB de archivos.
Se utilizarán 175 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
```

4.2.- **Realizar pruebas de rendimiento**

4.2.1.- <u>HTTP/1.1</u>

- Para poder realizar las pruebas lo más reales posibles, vamos a tener "certificados" en las dos máquinas, por lo tanto, vamos a instalarlos igual que el paso anterior.
- Para estudiar el rendimiento de un servidor web apache, primero debemos tener instalado apache como ya hemos visto antes y después vamos a instalar "gnuplot" en una maquina con entorno grafico para poder ver las gráficas cuando:

```
root@debian-virtual:~# apt install gnuplot
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  adwaita-icon-theme aglfn at-spi2-core glib-networking glib-networking-common
  glib-networking-services gnuplot-data gnuplot-qt gsettings-desktop-schemas
  libatk-bridge2.0-0 libatspi2.0-0 libcairo-gobject2 libdouble-conversion1
  libegl1-mesa libepoxy0 libevdev2 libgbm1 libglew2.0 libglu1-mesa libgtk-3-0
  libgtk-3-bin libgtk-3-common libinput-bin libinput10 libjson-glib-1.0-0
  libjson-glib-1.0-common liblua5.1-0 libmtdev1 libnotify4 libpcre16-3
  libproxy1v5 libqt5core5a libqt5dbus5 libqt5gui5 libqt5network5
  libqt5printsupport5 libqt5svg5 libqt5widgets5 librest-0.7-0
  libsoup-gnome2.4-1 libsoup2.4-1 libwacom-bin libwacom-common libwacom2
  libwayland-client0 libwayland-cursor0 libwayland-egl1-mesa
  libwayland-server0 libwxbase3.0-0v5 libwxgtk3.0-0v5 libxcb-icccm4
  libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-util0 libxcb-util0
  libxcb-xfixes0 libxcb-xinerama0 libxcb-xkb1 libxkbcommon-x11-0 libxkbcommon0
  mesa-utils notification-daemon qt5-gtk-platformtheme qttranslations5-l10n
Se instalarán los siguientes paquetes NUEVOS:
  adwaita-icon-theme aglfn at-spi2-core glib-networking glib-networking-common
  glib-networking-services gnuplot gnuplot-data gnuplot-qt
  gsettings-desktop-schemas libatk-bridge2.0-0 libatspi2.0-0 libcairo-gobject2
  libdouble-conversion1 libegl1-mesa libepoxy0 libevdev2 libgbm1 libglew2.0
  libglu1-mesa libgtk-3-0 libgtk-3-bin libgtk-3-common libinput-bin libinput10
  libjson-glib-1.0-0 libjson-glib-1.0-common liblua5.1-0 libmtdev1 libnotify4
  libpcre16-3 libproxy1v5 libqt5core5a libqt5dbus5 libqt5gui5 libqt5network5
  libqt5printsupport5 libqt5svg5 libqt5widgets5 librest-0.7-0
  libsoup-gnome2.4-1 libsoup2.4-1 libwacom-bin libwacom-common libwacom2
  libwayland-client0 libwayland-cursor0 libwayland-egl1-mesa
  libwayland-server0 libwxbase3.0-0v5 libwxgtk3.0-0v5 libxcb-icccm4
  libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-util0 libxcb-util0
  libxcb-xfixes0 libxcb-xinerama0 libxcb-xkb1 libxkbcommon-x11-0 libxkbcommon0
  mesa-utils notification-daemon qt5-gtk-platformtheme qttranslations5-l10n
```

```
0 actualizados, 66 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 37,4 MB de archivos.
Se utilizarán 141 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
```

- Una vez instalado vamos a utilizar “apache benmark” Lo que haremos será enviar un número determinado de peticiones (n) en grupos de varias (de “n” en “n”) a un sitio determinado. El resultado lo guardaremos en un archivo .csv ([Nombre_Fichero].csv) para luego procesarlo con GNUPloit. Que tendrá la siguiente estructura:

```
ab -g resultados.csv -n [Numero_Peticiones] -c [Grupo_Peticiones] http://[IP_NombrePagina]/
```

- Es importante añadir la última “/”.
- También es importante que el resultado del comando anterior se guarde en un fichero en este caso puede ser un “.csv”.

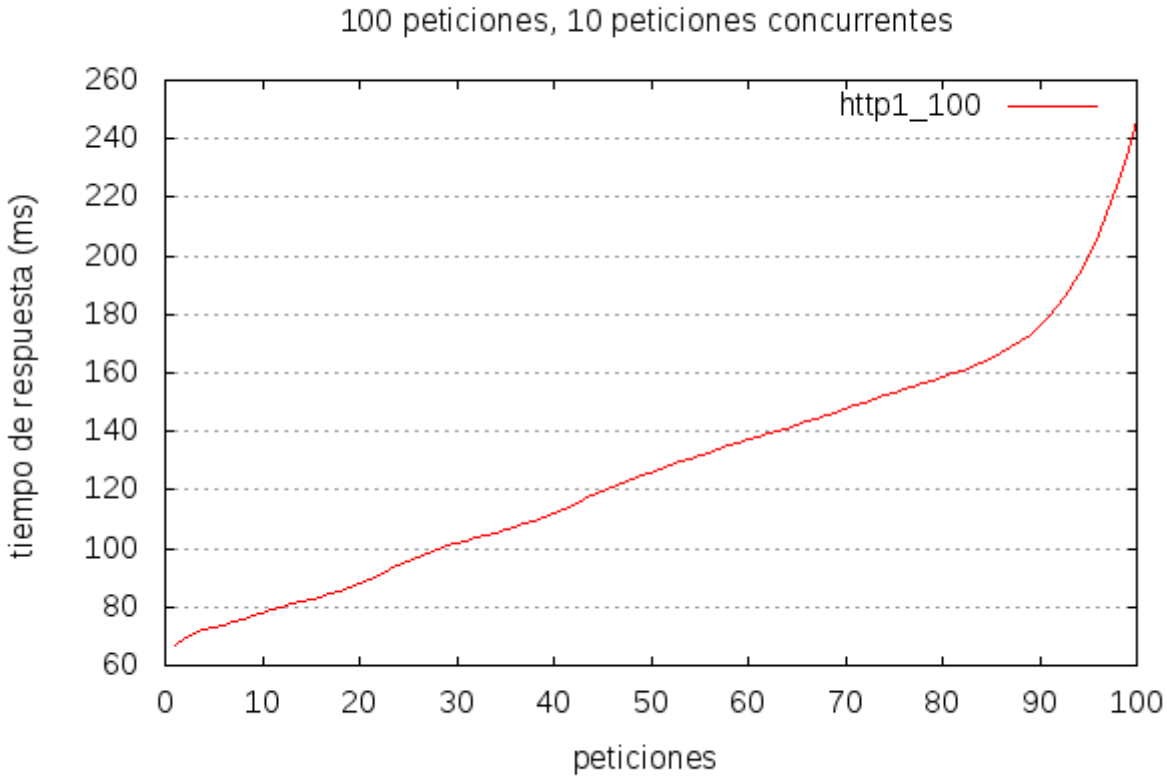
4.2.1.1.- <u>100 peticiones de 10 en 10.</u>

```
usuario@debian-virtual:~/Graficas$ ab -g http1_100.csv -n 100 -c 10 https://192.168.1.73/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.73 (be patient).....done



Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.73
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   1.307 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      1097500 bytes
HTML transferred:       1070100 bytes
Requests per second:    76.53 [#/sec] (mean)
Time per request:       130.666 [ms] (mean)
Time per request:       13.067 [ms] (mean, across all concurrent requests)
Transfer rate:          820.24 [Kbytes/sec] received

Connection Times (ms)
```

```
           min  mean[+/-sd] median   max
Connect:      53   93  28.2     85     211
Processing:    2   35  30.0     29     108
Waiting:       1   22  17.0     26      58
Total:        67  128  39.4    128     246

Percentage of the requests served within a certain time (ms)
  50%    128
  66%    143
  75%    154
  80%    160
  90%    181
  95%    207
  98%    230
  99%    246
 100%    246 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http1_100.png"
#Titulo Grafica
set title "100 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http1_100.csv" using 9 smooth sbezier with lines title "http1_100"
```



4.2.1.2.- 500 peticiones de 10 en 10

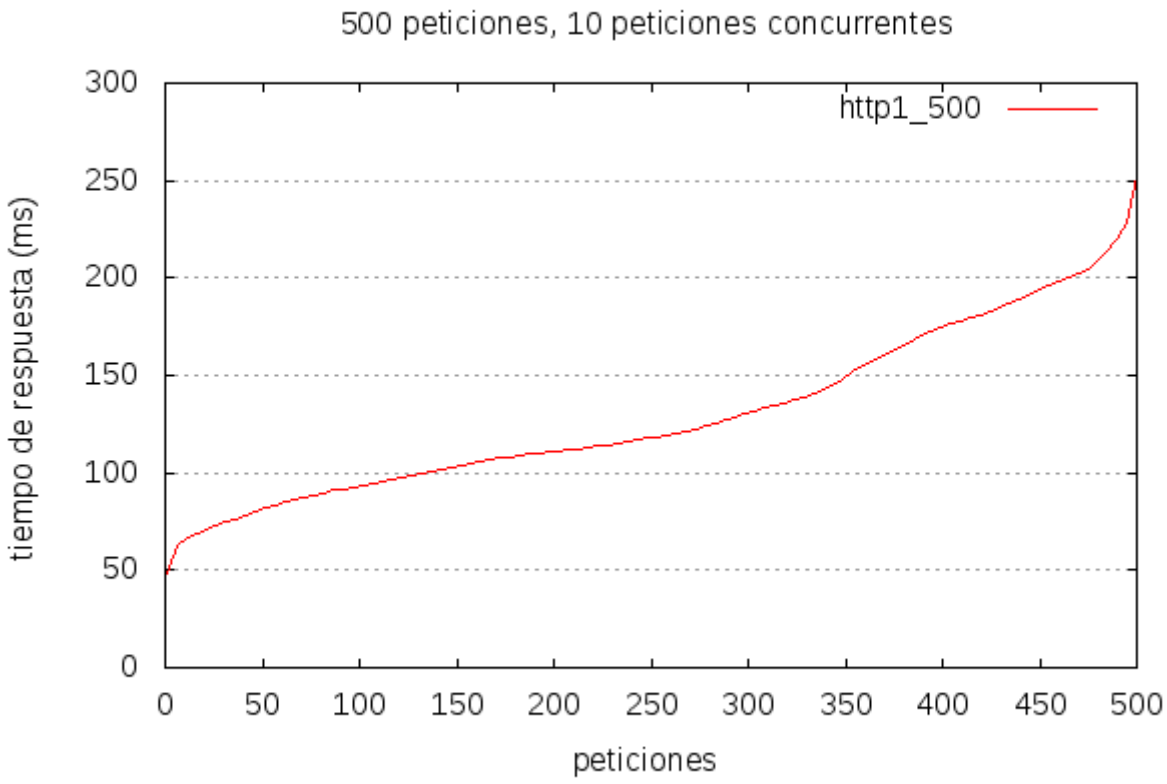
```
usuario@debian-virtual:~/Graficas$ ab -g http1_500.csv -n 500 -c 10 https://192.168.1.73/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.73 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.73
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256
```

```
Document Path:        /
Document Length:      10701 bytes

Concurrency Level:    10
Time taken for tests: 6.520 seconds
Complete requests:    500
Failed requests:      0
Total transferred:    5487500 bytes
HTML transferred:     5350500 bytes
Requests per second:  76.69 [#/sec] (mean)
Time per request:     130.396 [ms] (mean)
Time per request:     13.040 [ms] (mean, across all concurrent requests)
Transfer rate:        821.94 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       34   95  26.8     92     232
Processing:     1   35  32.3     25     123
Waiting:        1   24  20.9     20      98
Total:         48  130  42.1    118     253

Percentage of the requests served within a certain time (ms)
  50%    118
  66%    139
  75%    163
  80%    176
  90%    195
  95%    204
  98%    221
  99%    229
 100%    253 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http1_500.png"
#Titulo Grafica
set title "500 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
```

```
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http1_500.csv" using 9 smooth sbezier with lines title "http1_500"
```



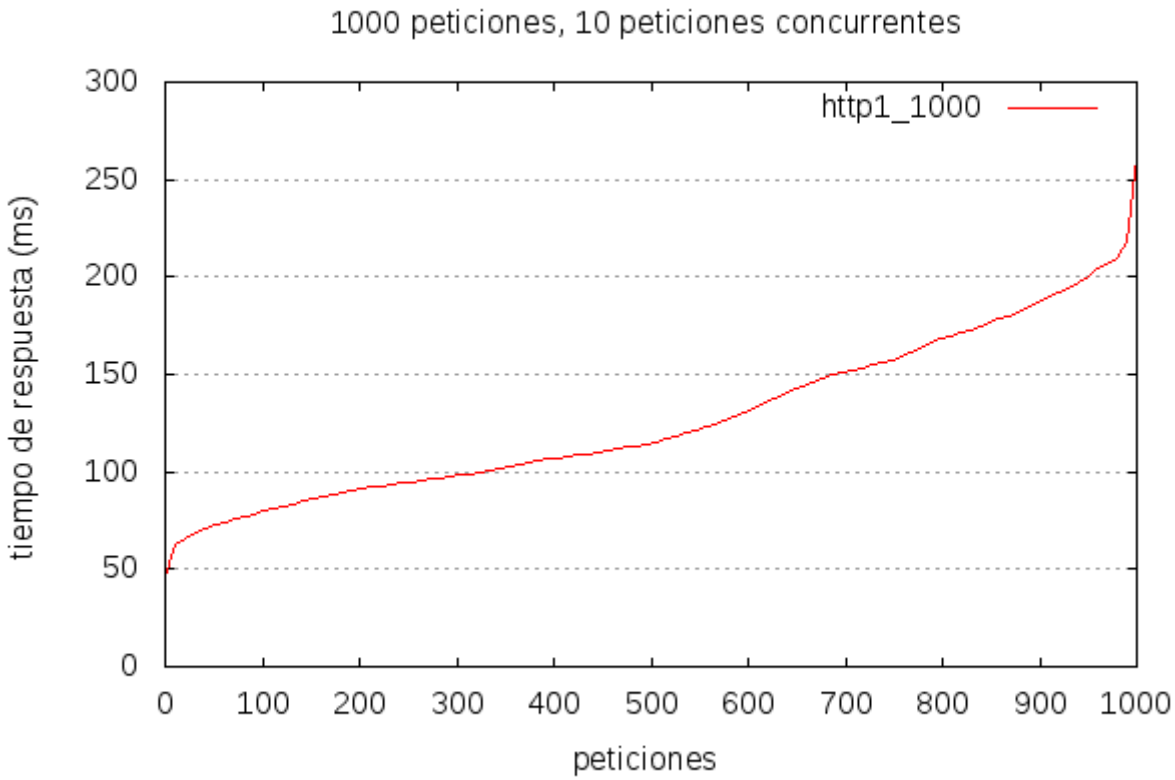
4.2.1.3.- 1000 peticiones de 10 en 10

```
usuario@debian-virtual:~/Graficas$ ab -g http1_1000.csv -n 1000 -c 10 https://192.168.1.73/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.73 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
```

```
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.73
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   12.727 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      10975000 bytes
HTML transferred:       10701000 bytes
Requests per second:    78.57 [#/sec] (mean)
Time per request:       127.268 [ms] (mean)
Time per request:       12.727 [ms] (mean, across all concurrent requests)
Transfer rate:          842.14 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       28   92  27.4     90     218
Processing:     1   34  30.7     26     117
Waiting:        1   23  19.5     18      94
Total:         46  127  40.7    115     262

Percentage of the requests served within a certain time (ms)
  50%    115
  66%    145
  75%    157
  80%    169
  90%    188
  95%    201
  98%    209
  99%    220
 100%    262 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http1_1000.png"
#Titulo Grafica
set title "1000 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http1_1000.csv" using 9 smooth sbezier with lines title "http1_1000"
```



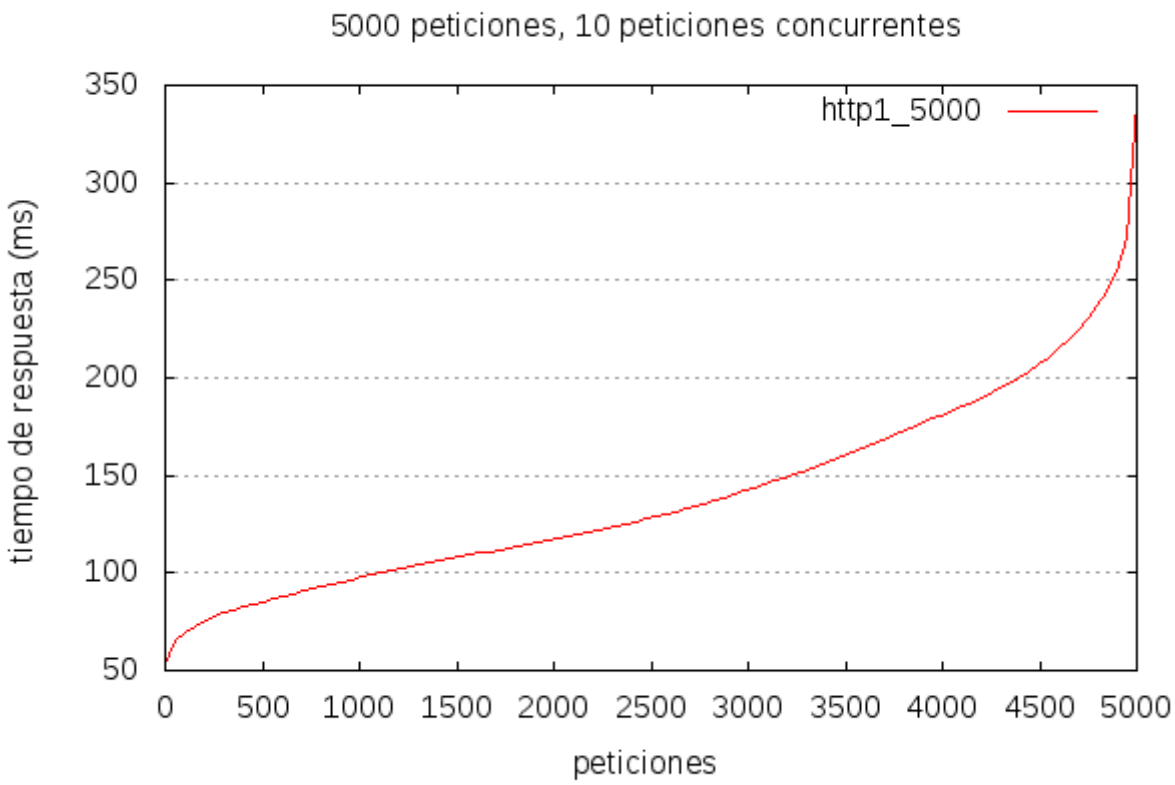
4.2.1.4.- 5000 peticiones de 10 en 10

```
usuario@debian-virtual:~/Graficas$ ab -g http1_5000.csv -n 5000 -c 10 https://192.168.1.73/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.73 (be patient)
Completed 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.73
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   69.821 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      54875000 bytes
HTML transferred:       53505000 bytes
Requests per second:    71.61 [#/sec] (mean)
Time per request:       139.643 [ms] (mean)
Time per request:       13.964 [ms] (mean, across all concurrent requests)
Transfer rate:          767.51 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       31  102  33.7     97     304
Processing:     1   38  34.3     27     181
```

```
Waiting:        1   25  21.2     20     129
Total:         53  139  47.9    128     342

Percentage of the requests served within a certain time (ms)
  50%    128
  66%    153
  75%    170
  80%    181
  90%    208
  95%    231
  98%    254
  99%    272
 100%    342 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http1_5000.png"
#Titulo Grafica
set title "5000 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http1_5000.csv" using 9 smooth sbezier with lines title "http1_5000"
```



4.2.1.5.- 10000 peticiones de 10 en 10

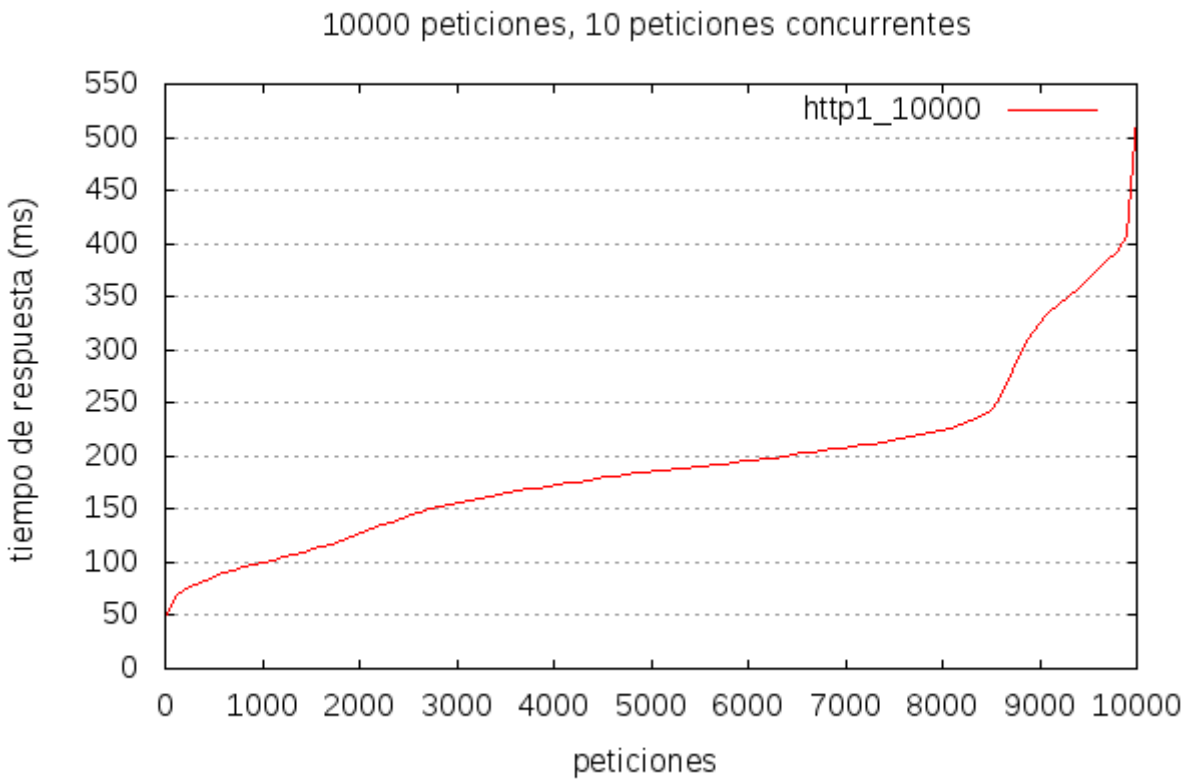
```
usuario@debian-virtual:~/Graficas$ ab -g http1_10000.csv -n 10000 -c 10 https://192.168.1.73/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.73 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests
```

```
Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.73
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   192.396 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      109750000 bytes
HTML transferred:       107010000 bytes
Requests per second:    51.98 [#/sec] (mean)
Time per request:       192.396 [ms] (mean)
Time per request:       19.240 [ms] (mean, across all concurrent requests)
Transfer rate:          557.07 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       25  152  62.8    154     515
Processing:     1   40  44.0     25     228
Waiting:        0   26  26.3     17     201
Total:         48  192  77.5    185     519

Percentage of the requests served within a certain time (ms)
  50%    185
  66%    203
  75%    215
  80%    224
  90%    325
  95%    367
  98%    393
  99%    407
 100%    519 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http1_10000.png"
#Titulo Grafica
set title "10000 peticiones, 10 peticiones concurrentes"
```

```
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http1_10000.csv" using 9 smooth sbezier with lines title "http1_10000"
```



4.2.2.- HTTP/2.

- Como hemos podido ver antes, ya tenemos operativo el protocolo http/2 en la maquina apache que tenemos dedicada para eso.
- Igual que en el punto anterior, una vez instalado vamos a utilizar "apache benmark" Lo que haremos será enviar un número determinado de peticiones (n) en grupos de varias (de "n" en "n") a un sitio determinado. El resultado lo guardaremos en un archivo .csv ([Nombre_Fichero].csv) para luego procesarlo con GNUPloit. Que tendrá la siguiente estructura:

```
ab -g resultados.csv -n [Numero_Peticiones] -c [Grupo_Peticiones] http://[IP_NombrePagina]/
```

- Es importante añadir la última "/".
- También es importante que el resultado del comando anterior se guarde en un fichero en este caso puede ser un ".csv".

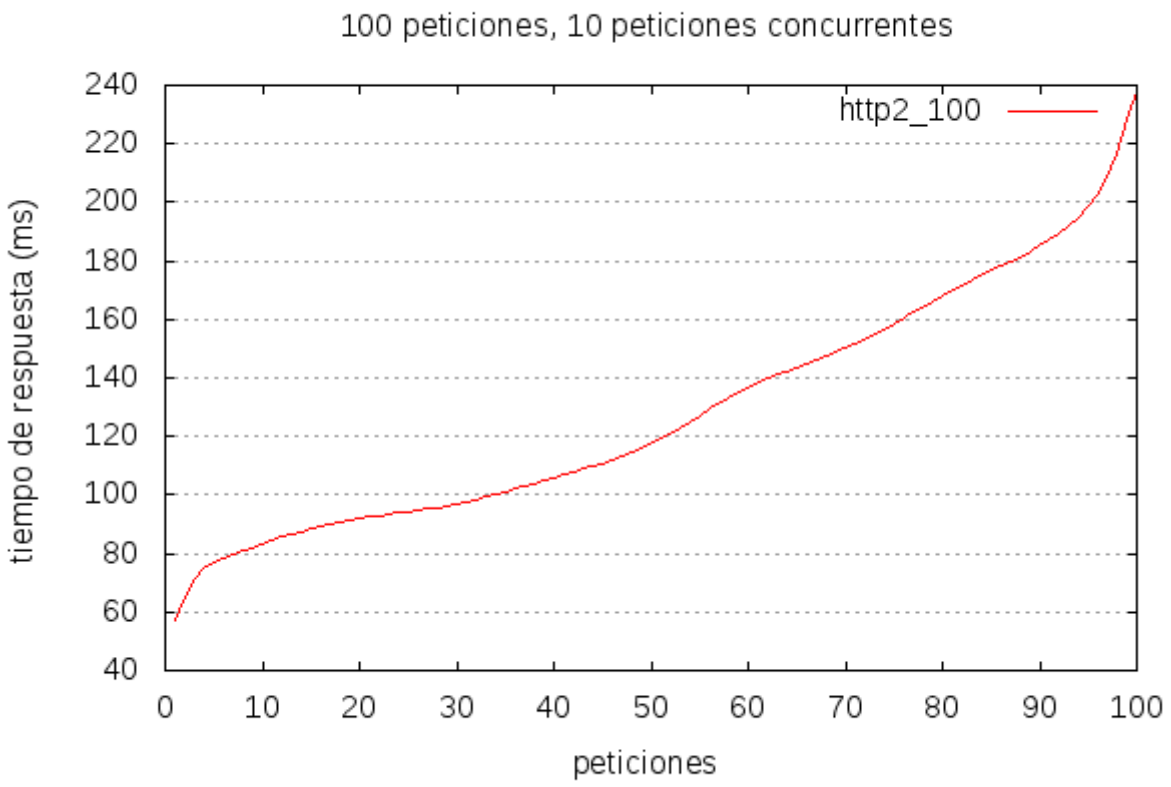
4.2.2.1.- 100 peticiones de 10 en 10.

```
usuario@debian-virtual:~/Graficas$ ab -g http2_100.csv -n 100 -c 10 https://192.168.1.139/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.139 (be patient).....done



Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.139
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   1.331 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      1099700 bytes
HTML transferred:       1070100 bytes
Requests per second:    75.14 [#/sec] (mean)
Time per request:       133.079 [ms] (mean)
Time per request:       13.308 [ms] (mean, across all concurrent requests)
Transfer rate:          806.98 [Kbytes/sec] received

Connection Times (ms)
```

```
           min  mean[+/-sd] median   max
Connect:      45   92  24.9     89     191
Processing:    2   37  33.8     26     114
Waiting:       1   25  20.1     21      74
Total:        57  129  40.7    116     238

Percentage of the requests served within a certain time (ms)
  50%    116
  66%    146
  75%    163
  80%    167
  90%    186
  95%    203
  98%    237
  99%    238
 100%    238 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http2_100.png"
#Titulo Grafica
set title "100 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http2_100.csv" using 9 smooth sbezier with lines title "http2_100"
```



4.2.2.2.- 500 peticiones de 10 en 10

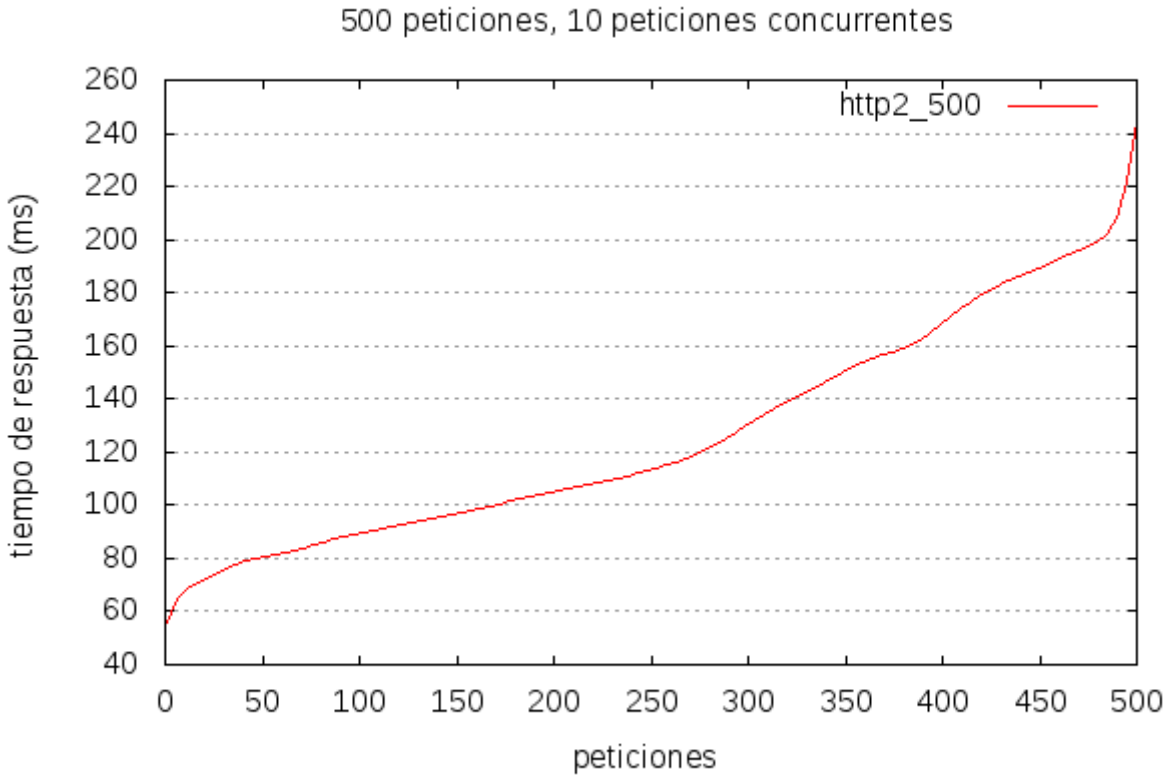
```
usuario@debian-virtual:~/Graficas$ ab -g http2_500.csv -n 500 -c 10 https://192.168.1.139/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.139 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.139
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256
```

```
Document Path:        /
Document Length:      10701 bytes

Concurrency Level:    10
Time taken for tests:  6.363 seconds
Complete requests:    500
Failed requests:      0
Total transferred:    5498500 bytes
HTML transferred:     5350500 bytes
Requests per second:   78.58 [#/sec] (mean)
Time per request:     127.258 [ms] (mean)
Time per request:     12.726 [ms] (mean, across all concurrent requests)
Transfer rate:        843.90 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       31   92  26.3     89     208
Processing:     1   34  32.1     23     116
Waiting:        1   23  20.1     18      95
Total:         56  126  41.1    114     245

Percentage of the requests served within a certain time (ms)
  50%    114
  66%    142
  75%    158
  80%    170
  90%    189
  95%    197
  98%    208
  99%    226
 100%    245 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http2_500.png"
#Titulo Grafica
set title "500 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
```

```
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http2_500.csv" using 9 smooth sbezier with lines title "http2_500"
```



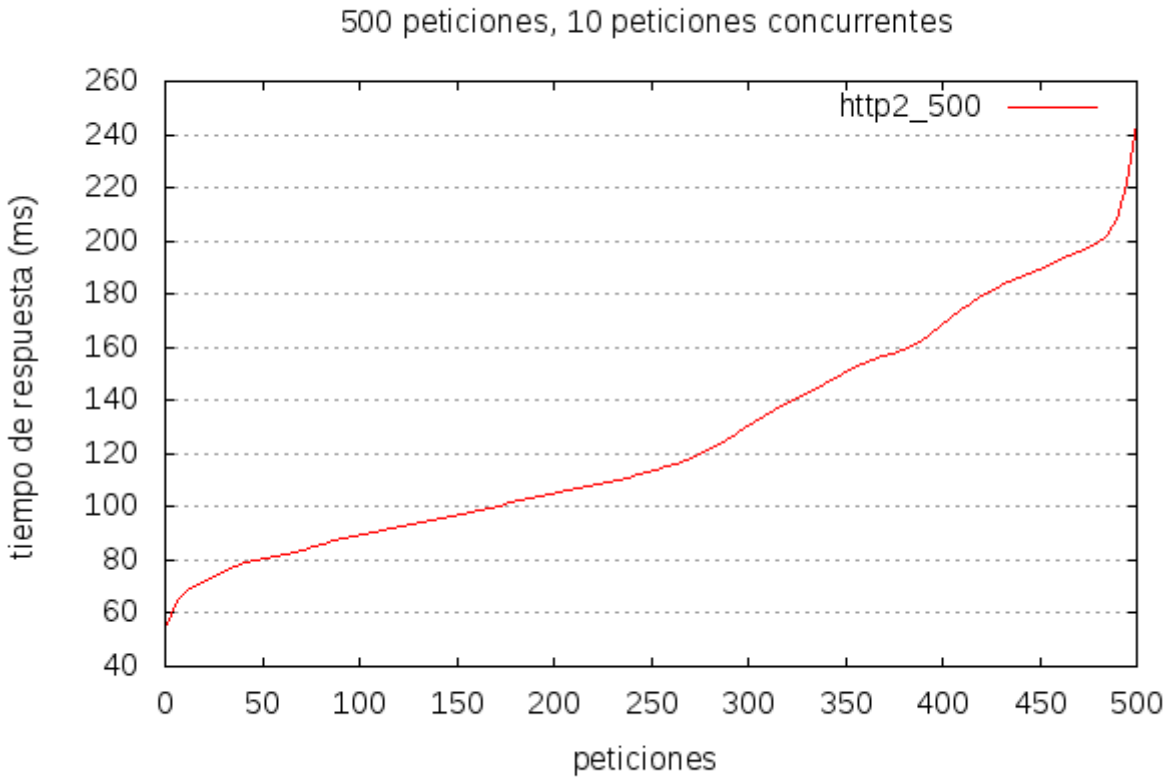
4.2.2.3.- 1000 peticiones de 10 en 10

```
usuario@debian-virtual:~/Graficas$ ab -g http2_1000.csv -n 1000 -c 10 https://192.168.1.139/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.139 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
```

```
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.139
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   12.627 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      10997000 bytes
HTML transferred:       10701000 bytes
Requests per second:    79.20 [#/sec] (mean)
Time per request:       126.269 [ms] (mean)
Time per request:       12.627 [ms] (mean, across all concurrent requests)
Transfer rate:          850.50 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       35   92  25.9     90     220
Processing:     1   33  31.4     21     117
Waiting:        0   23  20.4     18      95
Total:         41  125  40.1    116     249

Percentage of the requests served within a certain time (ms)
  50%    116
  66%    137
  75%    154
  80%    166
  90%    188
  95%    200
  98%    214
  99%    223
 100%    249 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http2_1000.png"
#Titulo Grafica
set title "1000 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http2_1000.csv" using 9 smooth sbezier with lines title "http2_1000"
```



4.2.2.4.- 5000 peticiones de 10 en 10

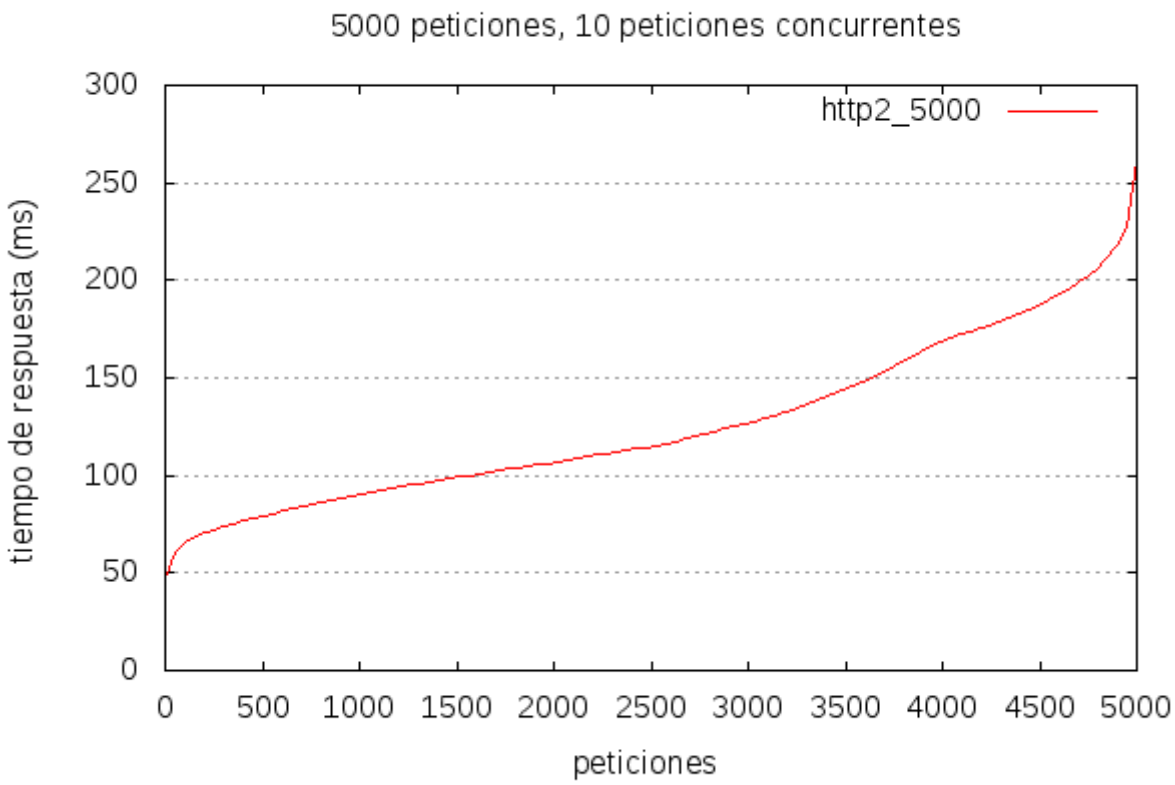
```
usuario@debian-virtual:~/Graficas$ ab -g http2_5000.csv -n 5000 -c 10 https://192.168.1.139/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
```

```
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.139 (be patient)
Completed 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.139
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   63.143 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      54985000 bytes
HTML transferred:       53505000 bytes
Requests per second:    79.19 [#/sec] (mean)
Time per request:       126.285 [ms] (mean)
Time per request:       12.629 [ms] (mean, across all concurrent requests)
Transfer rate:          850.40 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       26   92  26.8     90     248
Processing:     1   34  31.5     23     137
Waiting:        0   23  20.6     18     118
Total:         47  126  41.2    115     262

Percentage of the requests served within a certain time (ms)
```

```
 50%  115
 66%  137
 75%  156
 80%  169
 90%  187
 95%  203
 98%  218
 99%  227
100%  262 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http2_5000.png"
#Titulo Grafica
set title "5000 peticiones, 10 peticiones concurrentes"
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http2_5000.csv" using 9 smooth sbezier with lines title "http2_5000"
```



4.2.2.5.- 10000 peticiones de 10 en 10

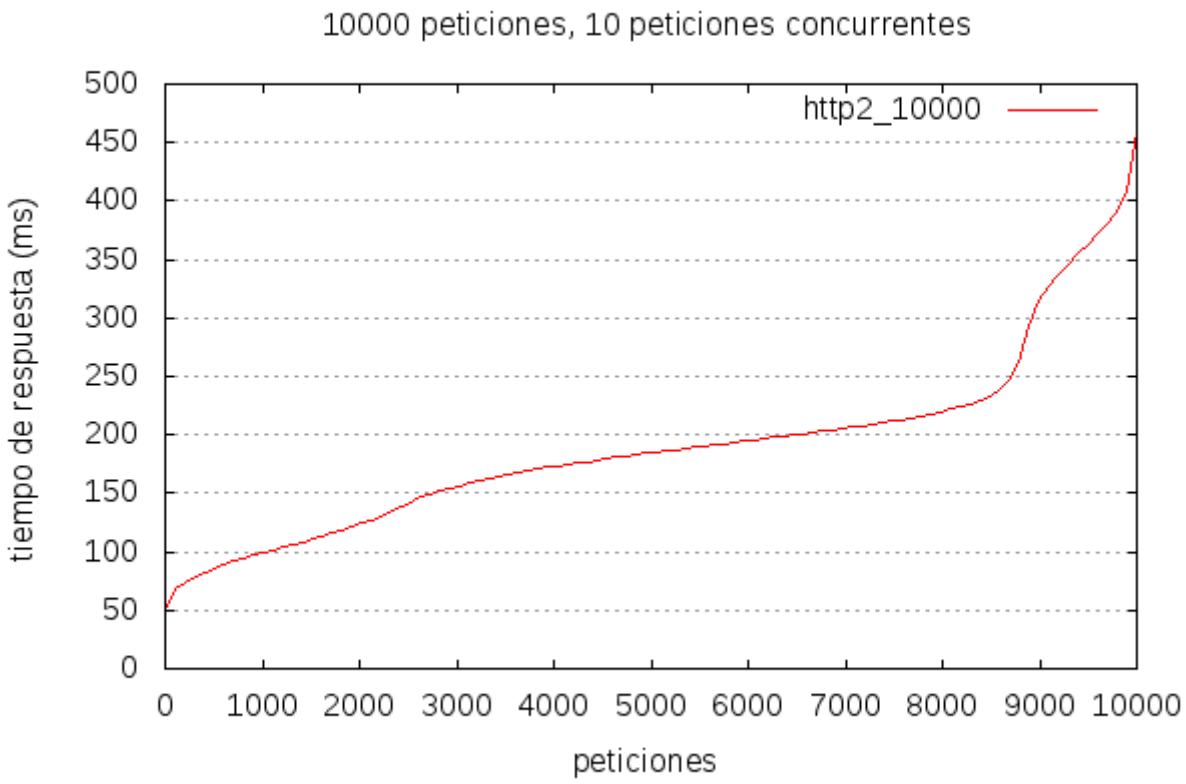
```
usuario@debian-virtual:~/Graficas$ ab -g http2_10000.csv -n 10000 -c 10 https://192.168.1.139/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.1.139 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests
```

```
Server Software:        Apache/2.4.25
Server Hostname:        192.168.1.139
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,1024,256

Document Path:          /
Document Length:        10701 bytes

Concurrency Level:      10
Time taken for tests:   189.781 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      109970000 bytes
HTML transferred:       107010000 bytes
Requests per second:    52.69 [#/sec] (mean)
Time per request:       189.781 [ms] (mean)
Time per request:       18.978 [ms] (mean, across all concurrent requests)
Transfer rate:          565.88 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       29  154  61.5    158     438
Processing:     1   36  42.4     17     225
Waiting:        0   25  26.3     11     180
Total:         49  189  75.6    185     461

Percentage of the requests served within a certain time (ms)
  50%    185
  66%    201
  75%    212
  80%    220
  90%    316
  95%    364
  98%    390
  99%    407
 100%    461 (longest request)
```

- Creamos la plantilla con la cual crearemos la gráfica:

```
#Tamaño imagen
set terminal png size 600
#Nombre imagen
set output "http2_10000.png"
#Titulo Grafica
set title "10000 peticiones, 10 peticiones concurrentes"
```

```
set size ratio 0.6
set grid y
#Nombre eje X
set xlabel "peticiones"
#Nombre eje Y
set ylabel "tiempo de respuesta (ms)"
#Nombre líneas grafica
plot "http2_10000.csv" using 9 smooth sbezier with lines title "http2_10000"
```

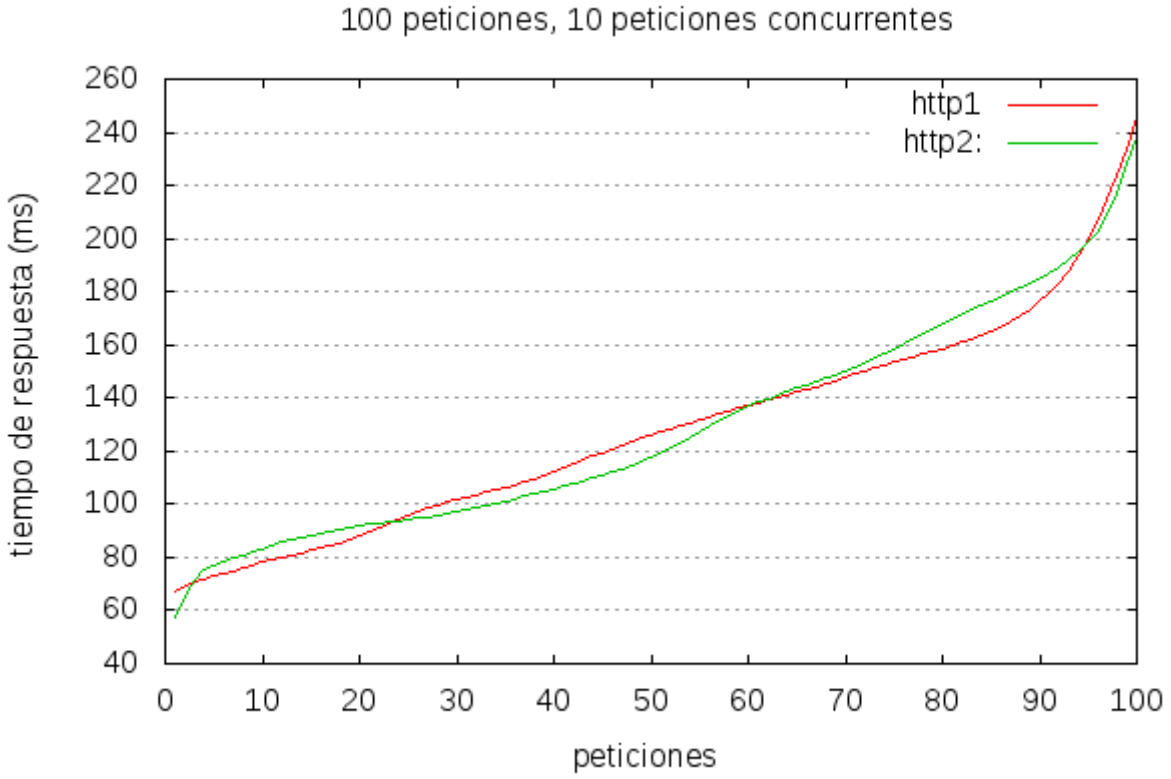


4.3.- **Graficas comparativas entre http/2 y http/1.1**

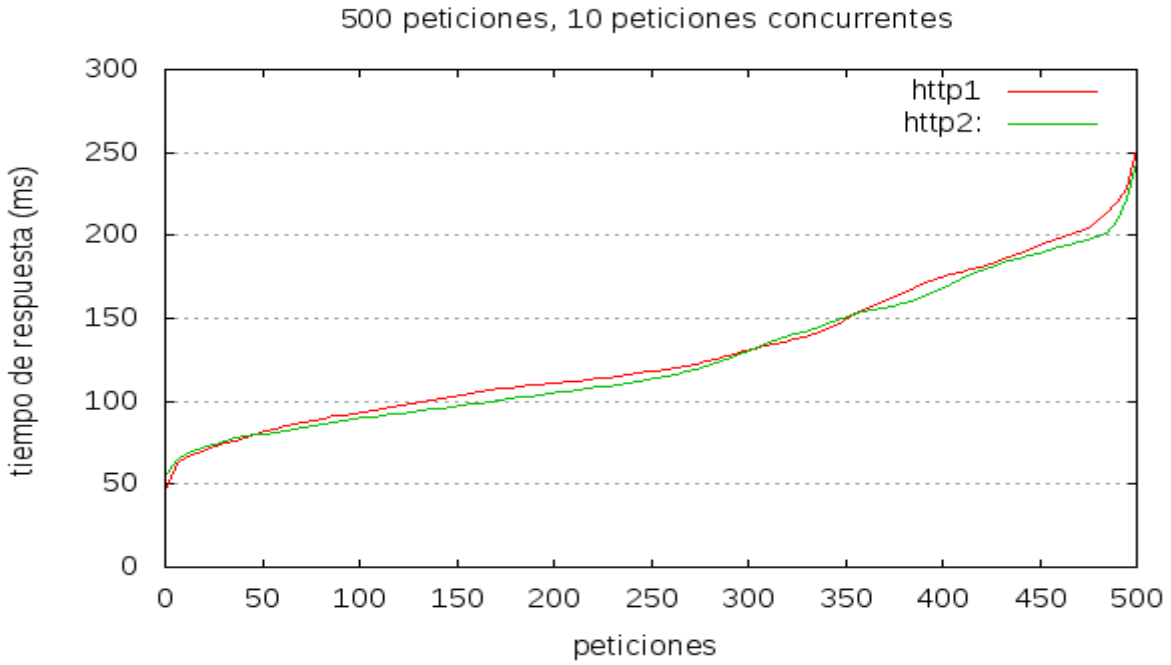
- Para realizar la graficas comparativas vamos a utilizar los estudios realizados anteriormente y los vamos a colocar en una gráfica donde podremos ver las dos líneas (http/1.1 y http/2).
- Para ello vamos a utilizar la siguiente estructura en la plantilla que después convertiremos en gráfica:

```
set terminal png size 600
set output "[Numero_Peticiones]http1_http2.png"
set title "[n] peticiones, [m] peticiones concurrentes"
set size ratio 0.6
set grid y
set xlabel "peticiones"
set ylabel "tiempo de respuesta (ms)"
plot "http1_[Numero_Peticiones].csv" using 9 smooth sbezier with lines title "http1", "http2_[Numero_Peticiones].csv" using 9 smooth sbezier with lines title "http2:"
```

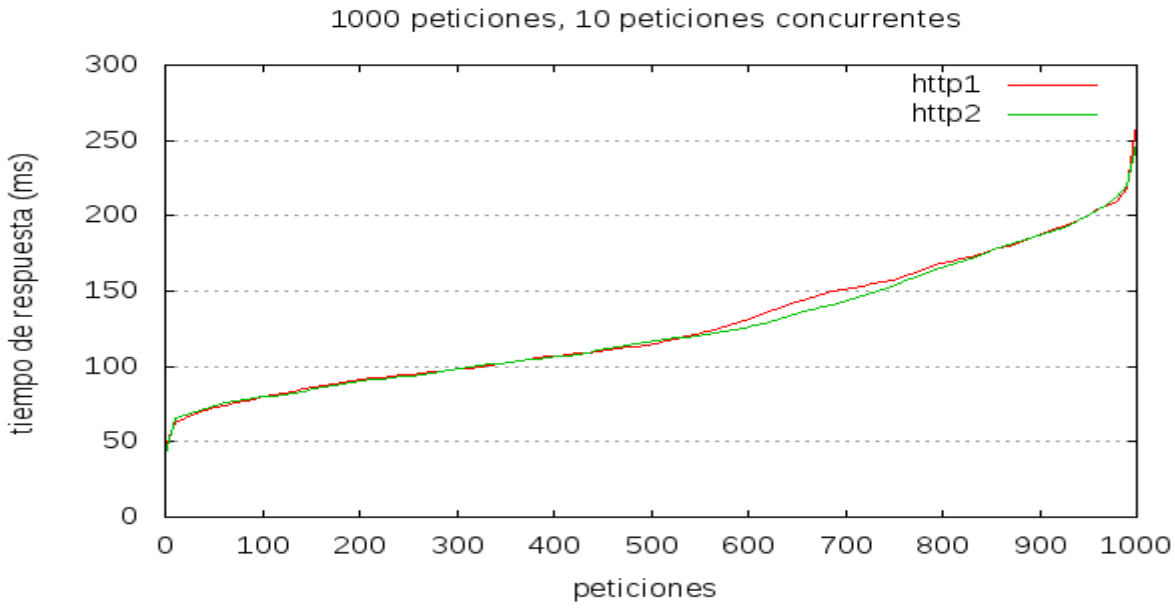
4.3.1.- **100 peticiones de 10 en 10**



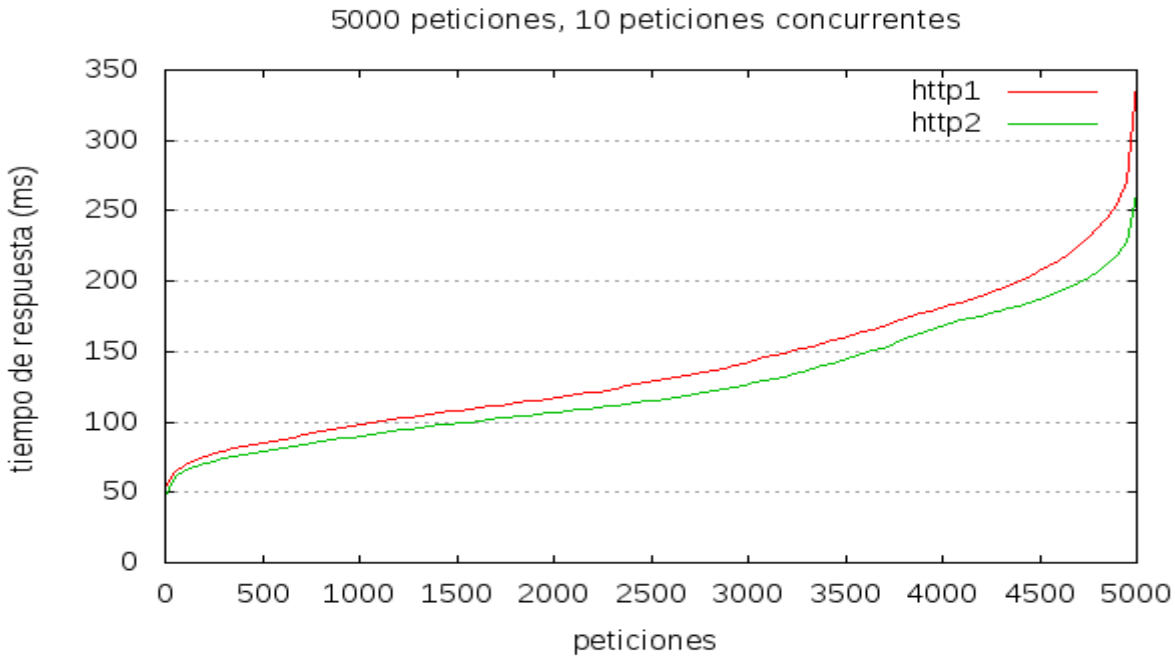
4.3.2.- 500 peticiones de 10 en 10



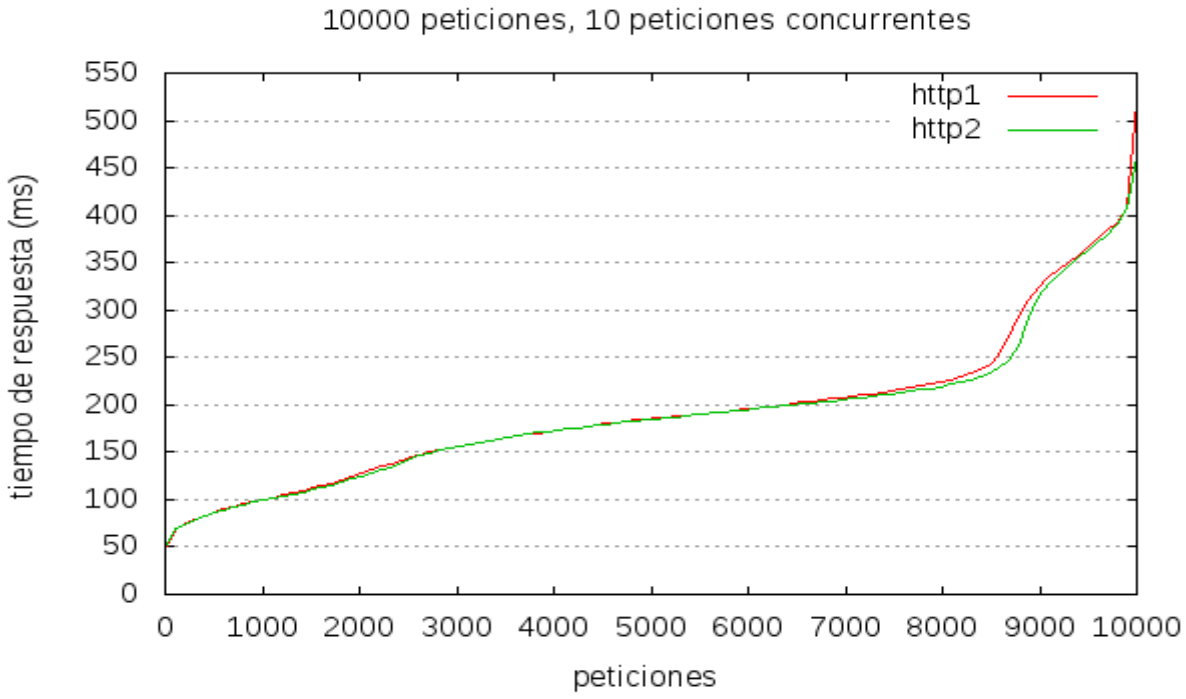
4.3.3.- 1000 peticiones de 10 en 10



4.3.4.- 5000 peticiones de 10 en 10



4.3.5.- 10000 peticiones de 10 en 10



5.- INSTALACIÓN DE UN CMS (WORDPRESS) Y PRUEBAS DE RENDIMIENTO CON GRÁFICAS.

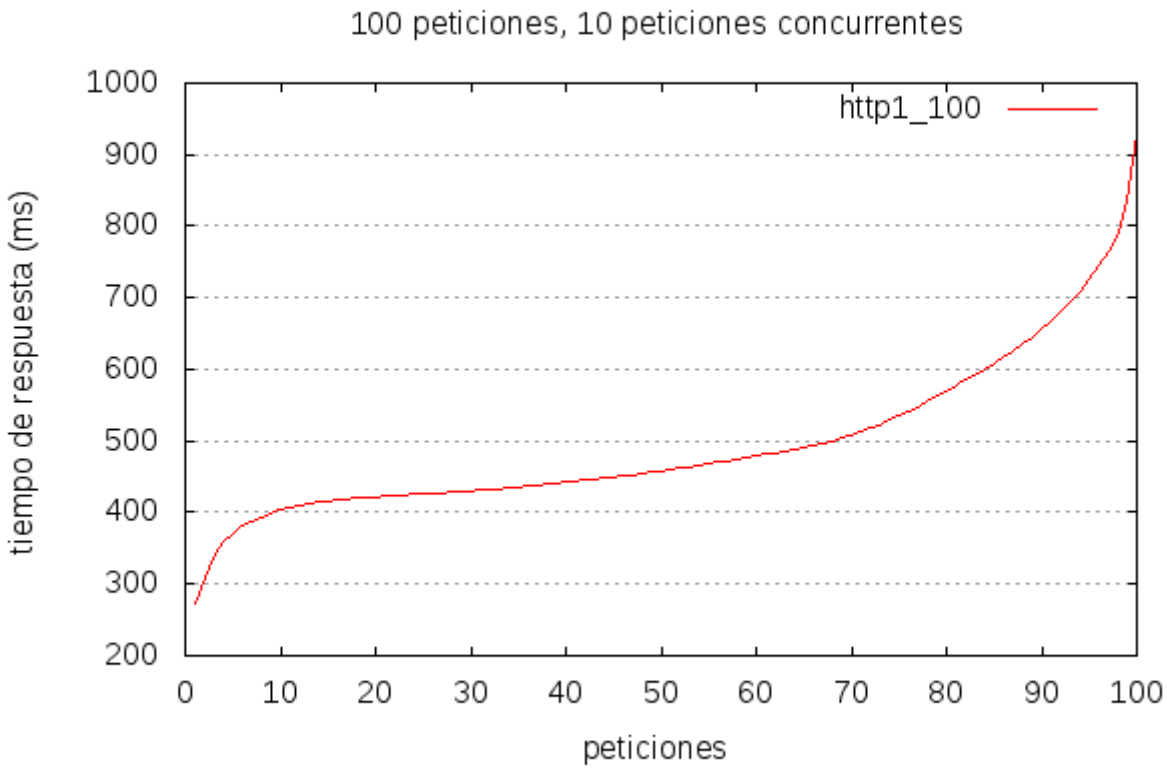
- Lo primero será descargar el cms en este caso será “Wordpress”, desde la página oficial en cada una de las máquinas que estamos utilizando, para ello vamos a utilizar el comando “wget”:

```
wget https://es-mx.wordpress.org/wordpress-4.7.5-es_MX.zip
```

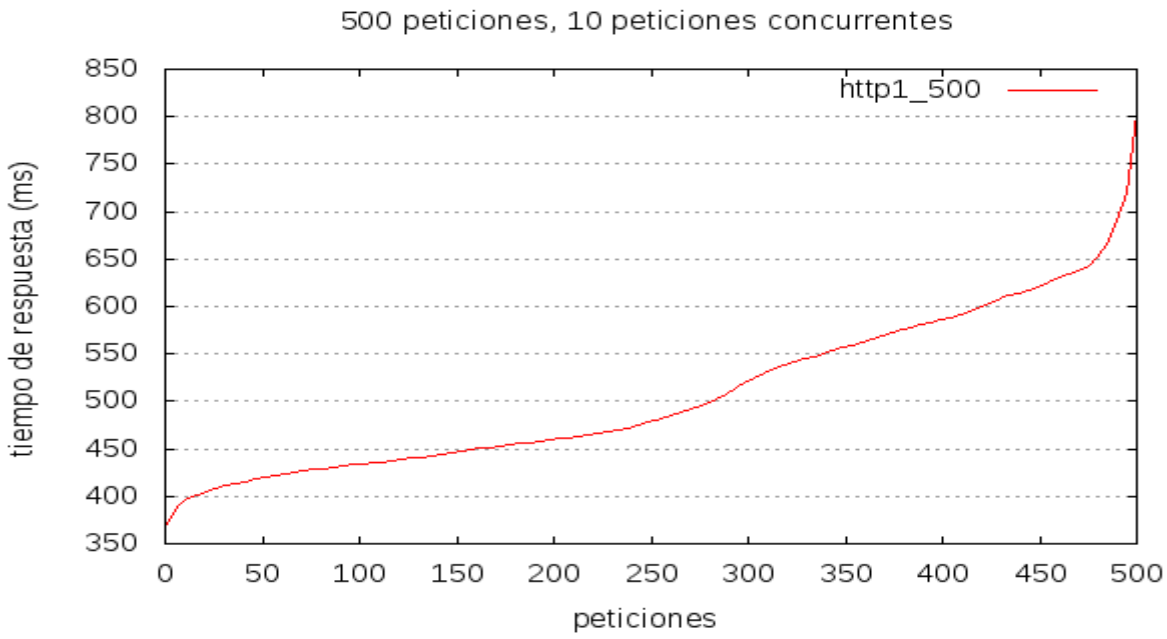
- Una vez instalado el CMS en ambas máquinas, procedemos a realizar las pruebas de rendimiento.

5.1.- **Graficas http/1.1**

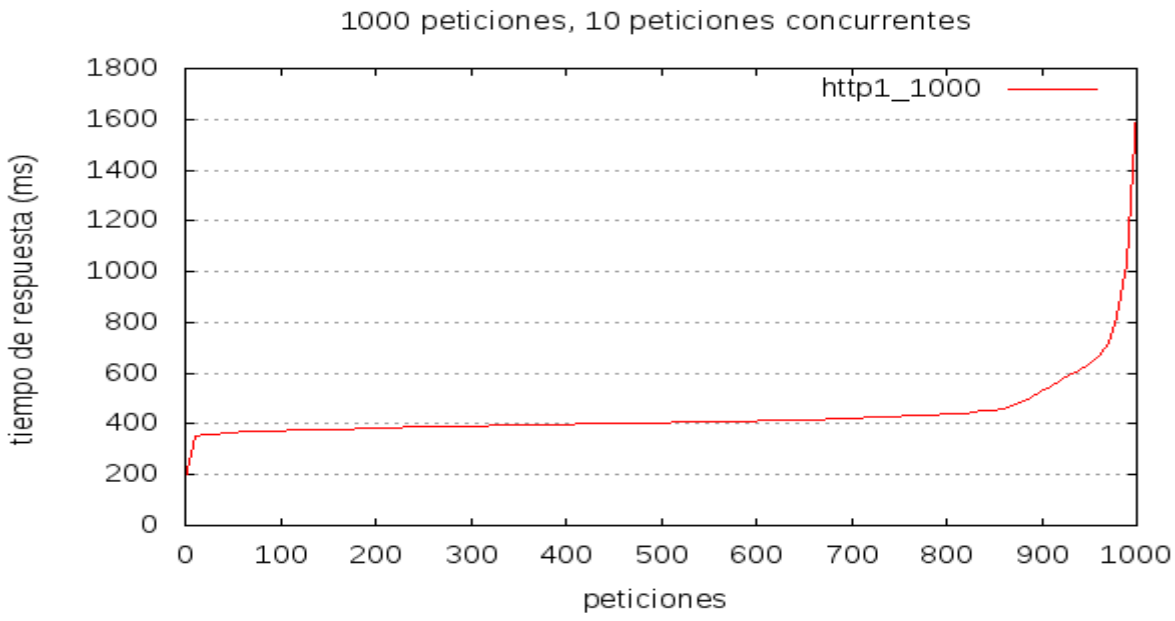
5.1.1.- **100 peticiones de 10 en 10**



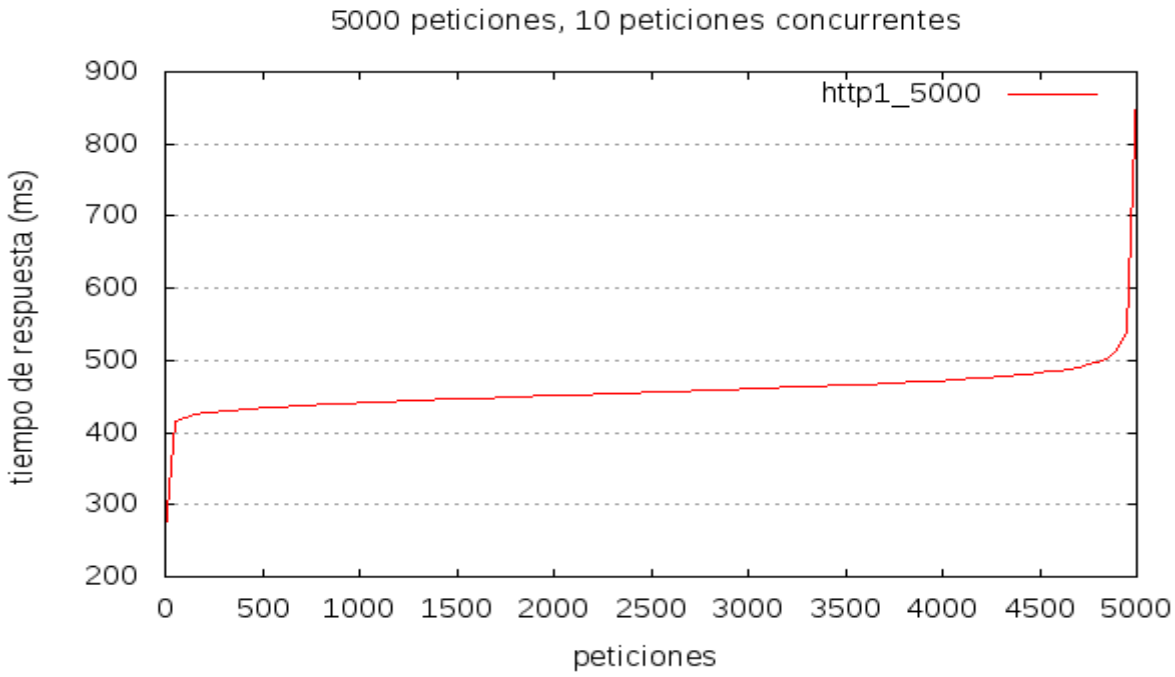
5.1.2.- 500 peticiones de 10 en 10



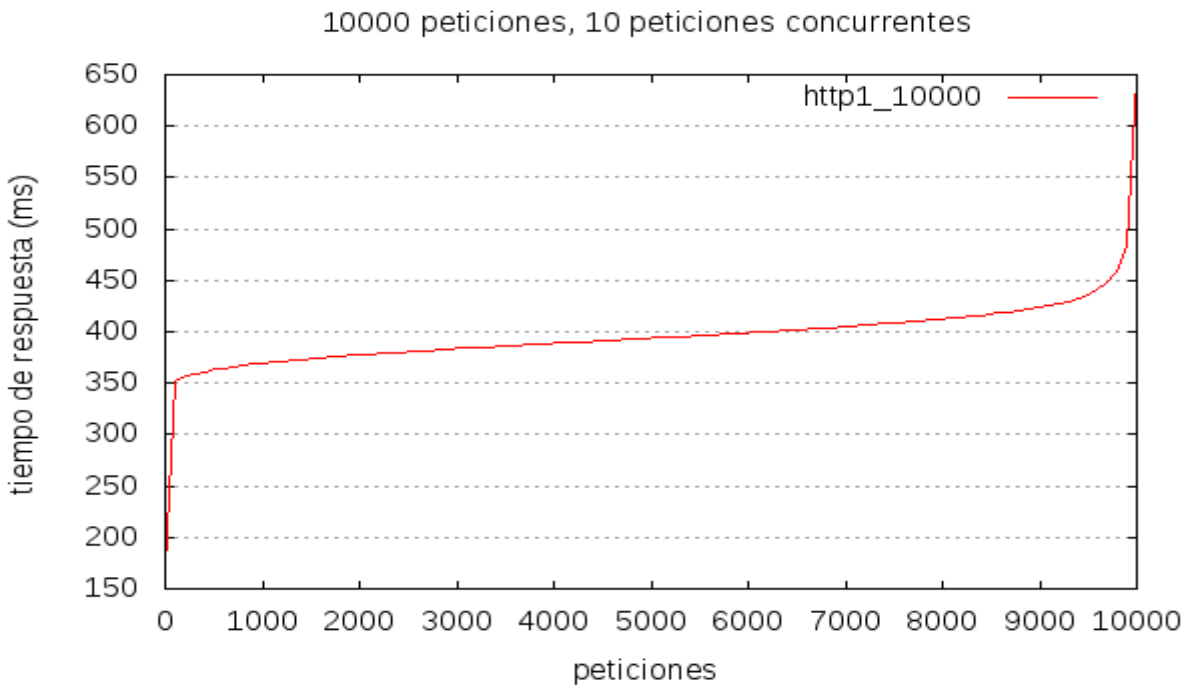
5.1.3.- 1000 peticiones de 10 en 10



5.1.4.- 5000 peticiones de 10 en 10

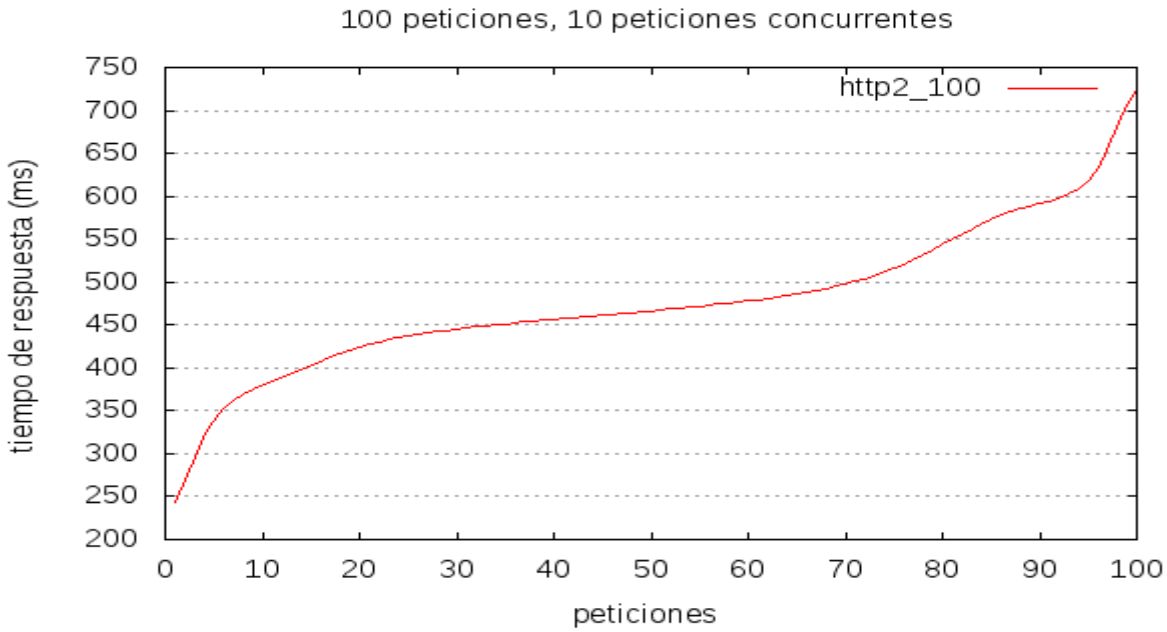


5.1.5.- 10000 peticiones de 10 en 10

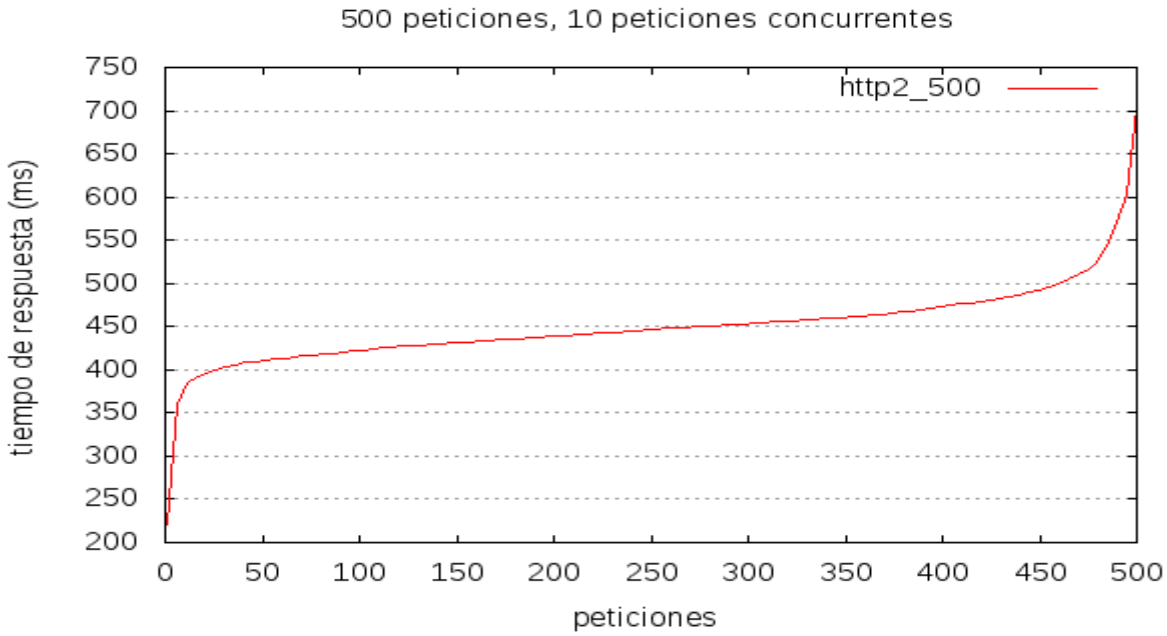


5.2.- **Http/2**

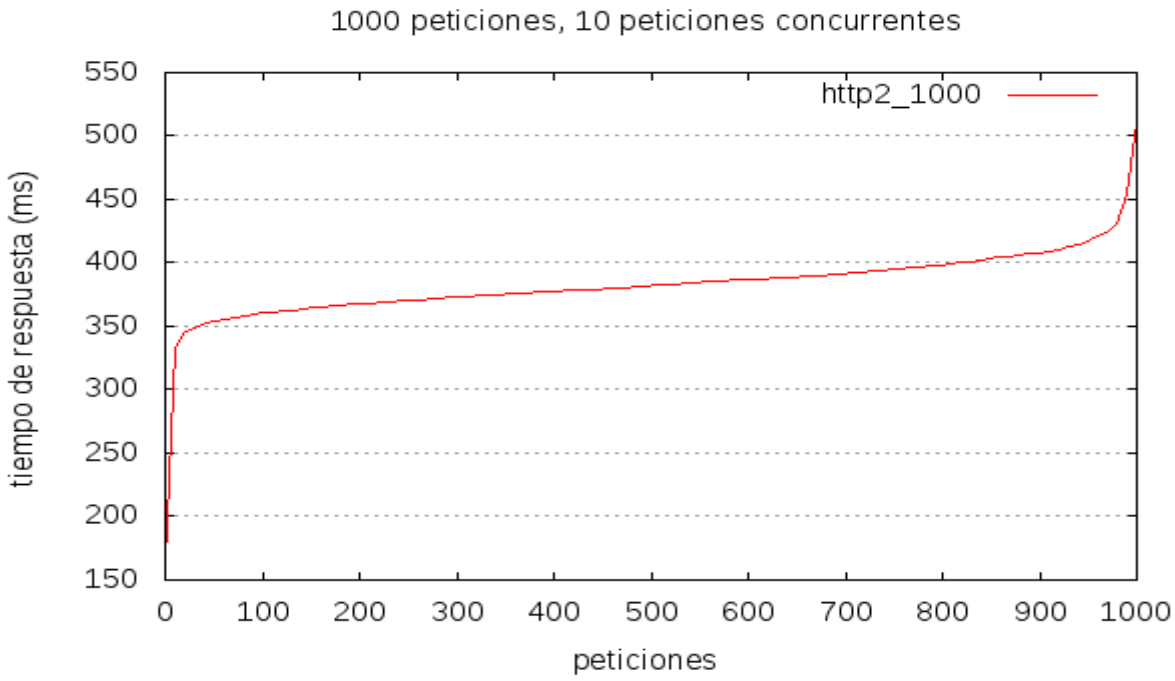
5.2.1.- <u>100 peticiones de 10 en 10</u>



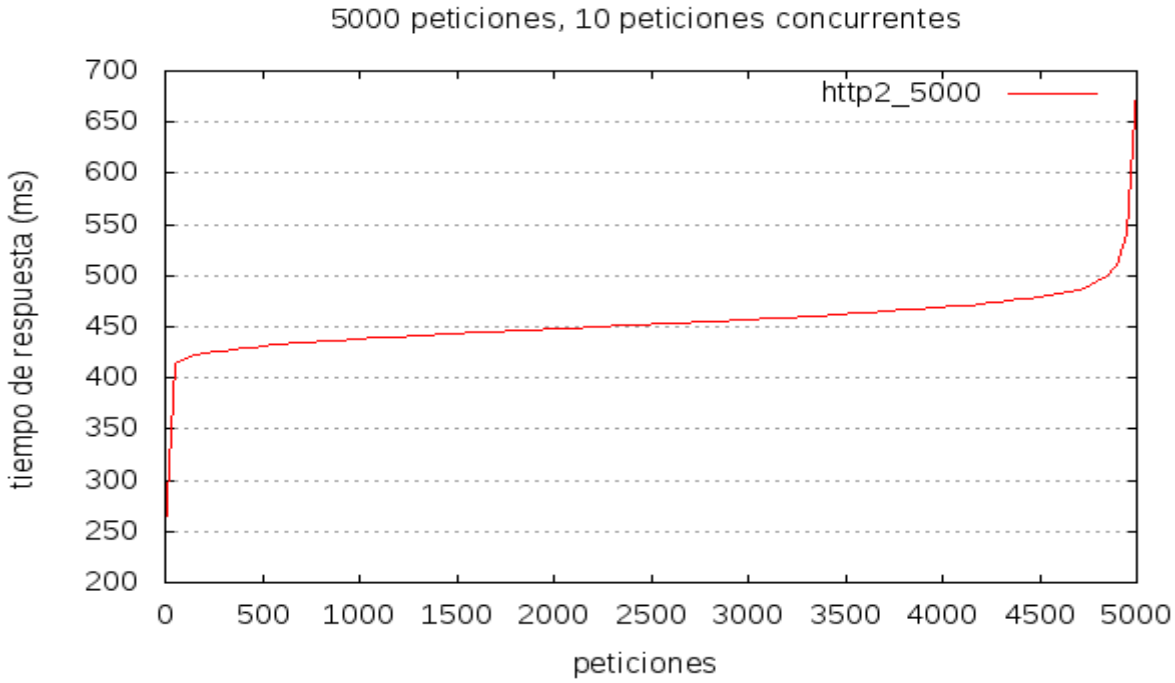
5.2.2.- <u>500 peticiones de 10 en 10</u>



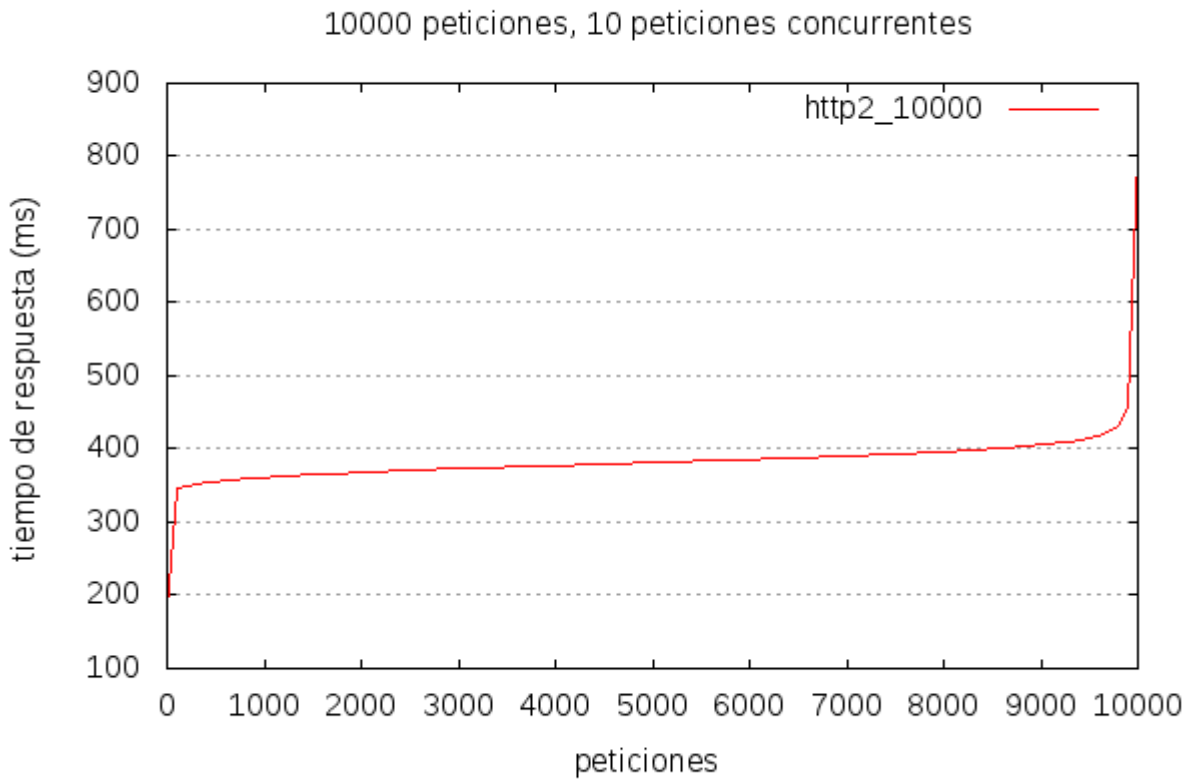
5.2.3.- 1000 peticiones de 10 en 10



5.2.4.- 5000 peticiones de 10 en 10



5.2.5.- 10000 peticiones de 10 en 10

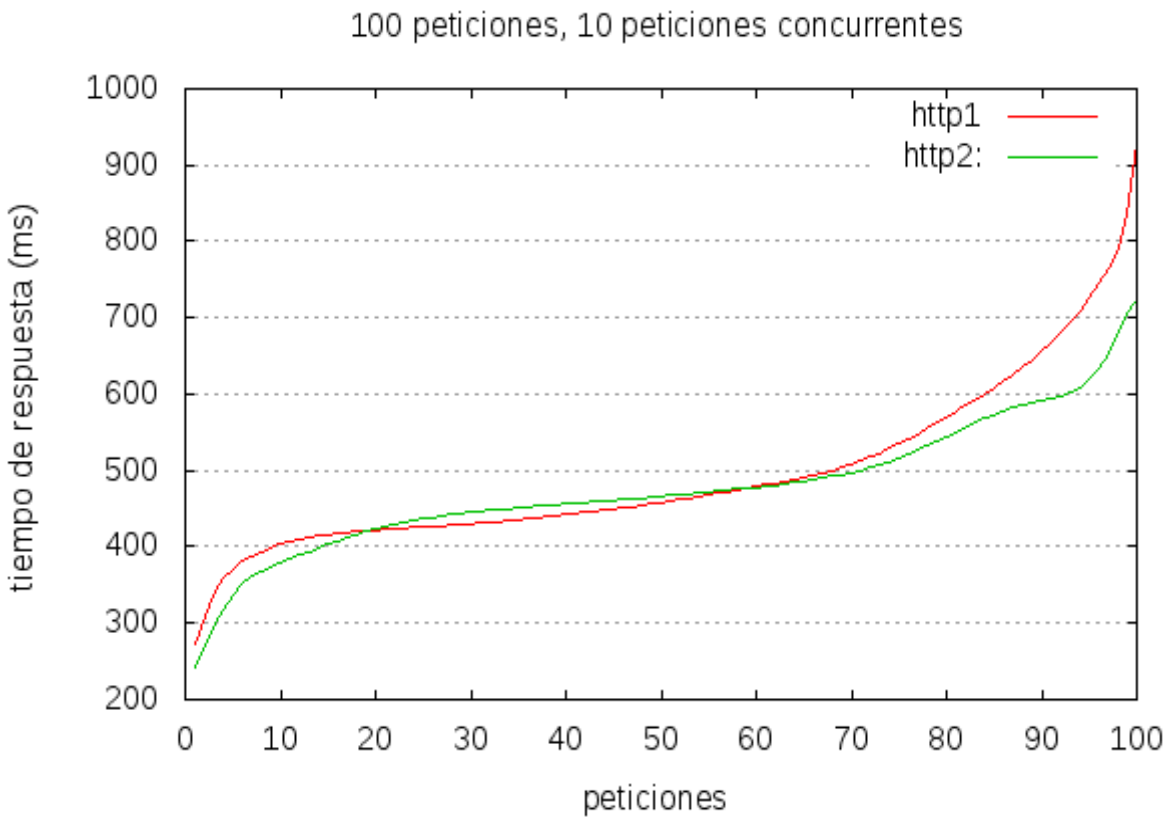


5.3.- **Graficas comparativas entre http/2 y http/1.1**

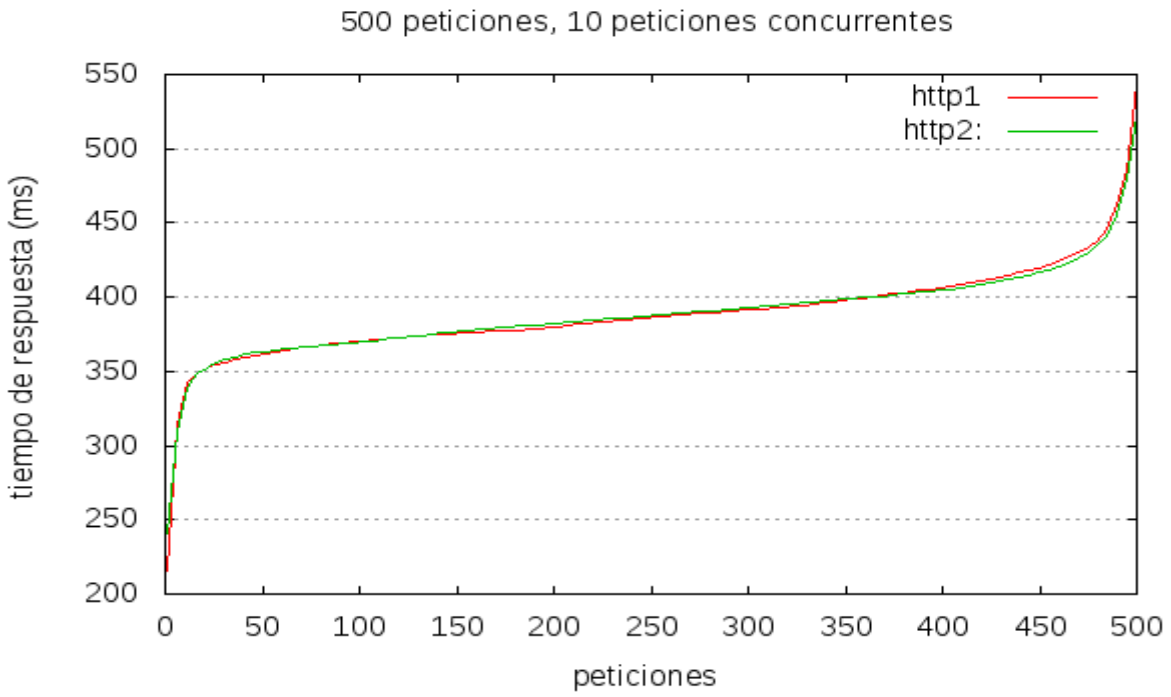
- Para realizar la graficas comparativas vamos a utilizar los estudios realizados anteriormente y los vamos a colocar en una gráfica donde podremos ver las dos líneas (http/1.1 y http/2).
- Para ello vamos a utilizar la siguiente estructura en la plantilla que después convertiremos en gráfica:

```
set terminal png size 600
set output "[Numero_Peticiones]http1_http2.png"
set title "[n] peticiones, [m] peticiones concurrentes"
set size ratio 0.6
set grid y
set xlabel "peticiones"
set ylabel "tiempo de respuesta (ms)"
plot "http1_[Numero_Peticiones].csv" using 9 smooth sbezier with lines title "http1", "http2_[Numero_Peticiones].csv" using 9 smooth sbezier with lines title "http2:"
```

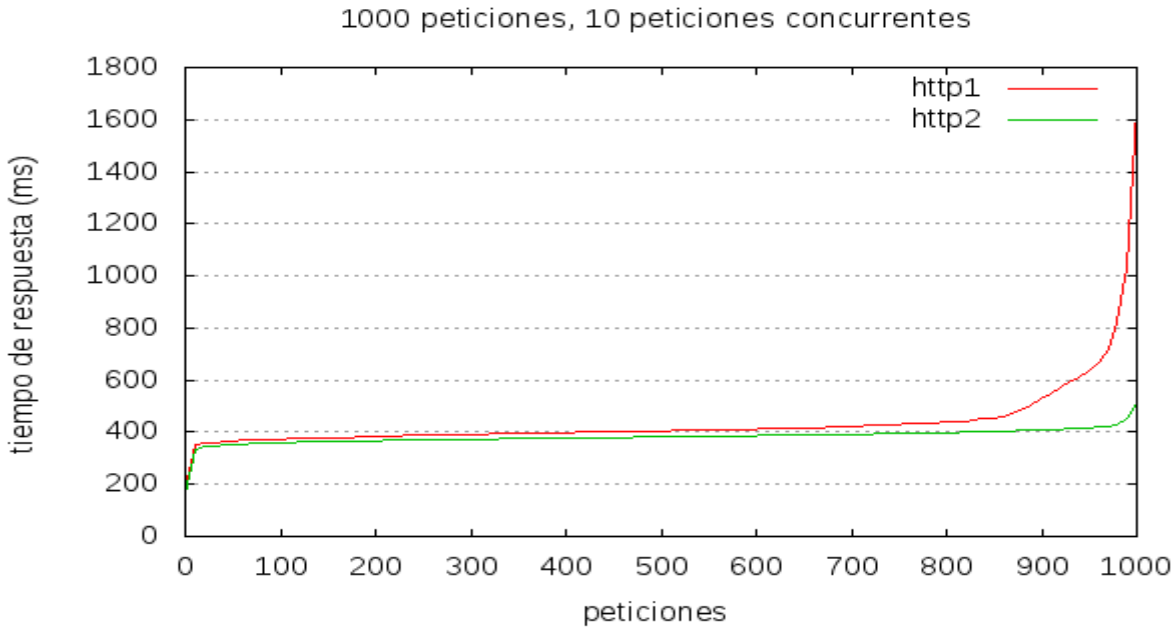
5.3.1.- **100 peticiones de 10 en 10**



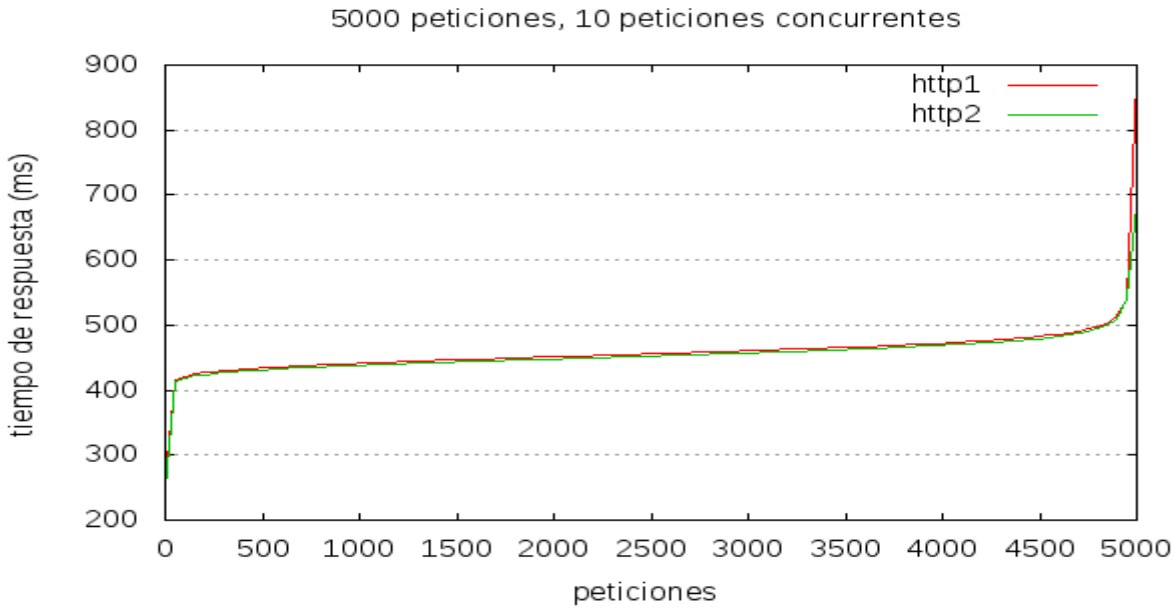
5.3.2.- 500 peticiones de 10 en 10



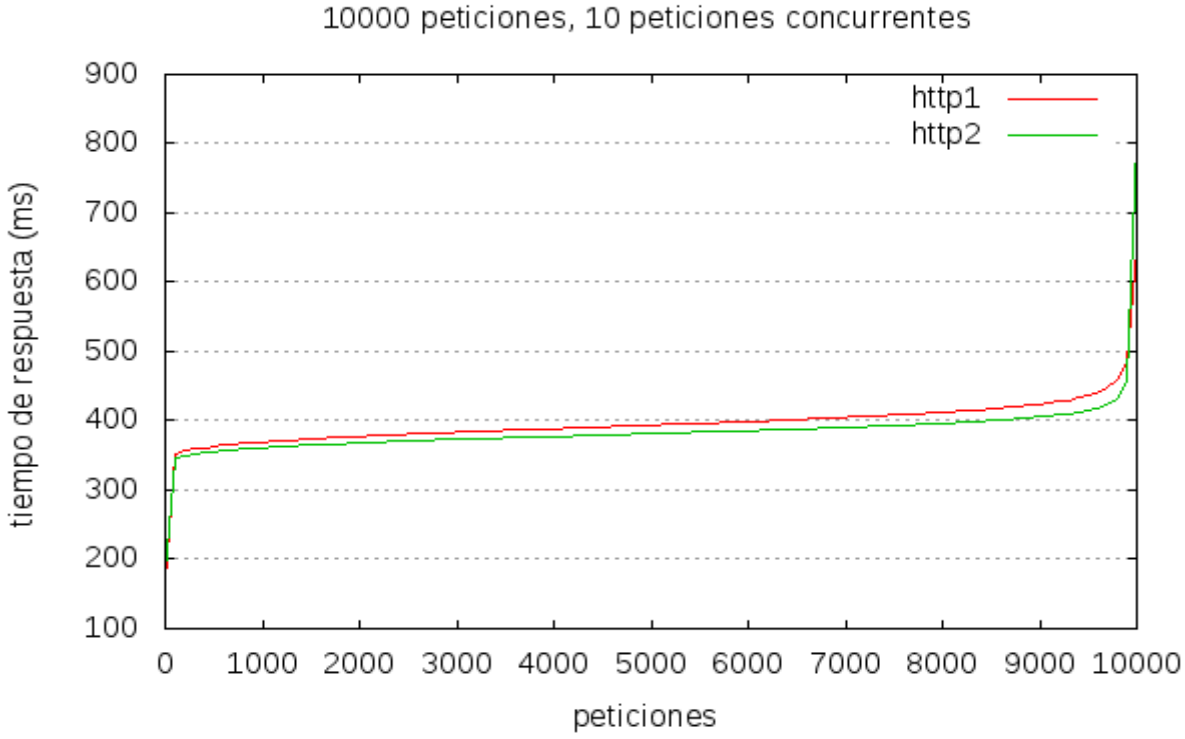
5.3.3.- 1000 peticiones de 10 en 10



5.3.4.- 5000 peticiones de 10 en 10



5.3.5.- 10000 peticiones de 10 en 10



5.4.- **Conclusión del estudio.**

- Podemos observar en el estudio realizado anteriormente, que la diferencia en peticiones pequeñas como son las peticiones estudiadas, no es tanta. También podemos observar que en el estudio realizado con una página estática como es la página por defecto de apache2, no es el mismo que con un “cms” (Wordpress), en el cual se nota más la diferencia.
- También podemos ver que en las gráficas en la cuales se llega a notar un poco más la diferencia, se ve como oscila al principio puede ser un poco más lento, pero después, conforme avanzan las peticiones se ve como es más rápido http/2.
- Por último, he sacado en conclusión que para poder configurar y poner en funcionamiento este protocolo es necesario la instalación de un certificado (SSL), dado que si no accedes con “https://”, no funciona dicho protocolo. Para poder realizar unas pruebas de rendimiento coherente he instalado certificados en las dos máquinas.

6.- CONCLUSION FINAL.

- Como hemos podido observar, el protocolo HTTP/2 es muy nuevo, encontrándose poco implantado ahora mismo en las páginas web más concurridas en este momento, paginas como “google” ya tiene implantado dicho protocolo, pero en cambio paginas como “Amazon” una de las páginas más importante en cuanto a compras online se refiere, no se encuentra en dicho protocolo.
- También hay que señalar como hemos visto en el estudio del rendimiento con gráficas, que no siempre se trata de una mejora muy considerable, todo depende del contenido que tenga dicha página, si se trata de una página estática no se notara tanta como en una página con un CMS o con JavaScript.
- Su navegador seguirá enviando peticiones a un servidor y obtendrá respuestas con lo necesario para para renderizar la página web como se debe, pero algunos matices por detrás van a cambiar.
- HTTP/2 trae nuevas características tales como:
    - Multiplexed streams
    - Server push
    - La compresión de HEADERS
    - El formato binario.
- El principal problema del protocolo HTTP/1.1 es que sólo permite una petición por cada conexión TCP. Para tratar de solucionar esto, los navegadores trataron de realizar varias peticiones en paralelo estableciendo varias conexiones TCP simultáneas, pero esto también es contraproducente porque se termina por congestionar la red.
- Resumiendo, se podría decir que si se hacen demasiadas peticiones HTTP/1.1, se termina por disminuir el rendimiento.