



http2 explained

Daniel Stenberg

Tabla de contenido

Introduction	1.1
Antecedentes	1.2
HTTP hoy	1.3
Estrategias para evitar los dolores de latencia	1.4
Actualizando HTTP	1.5
Conceptos de http2	1.6
El protocolo http2	1.7
Extensiones	1.8
Un mundo http2	1.9
http2 en Firefox	1.10
http2 en Chromium	1.11
http2 en curl	1.12
Después de http2	1.13
Otras lecturas	1.14
Agradecimientos	1.15

http2 explicado

Este es un documento detallado que describe HTTP/2 ([RFC 7540](#)), sus antecedentes, conceptos, el protocolo y algo sobre las implementaciones existentes y lo que nos puede deparar el futuro.

El sitio <http://daniel.haxx.se/http2/> es el home canónico de este proyecto.

En <https://github.com/bagder/http2-explained> se encuentra el código fuente de todo el contenido del libro.

CONTRIBUIR

Se alienta y da la bienvenida a cualquier ayuda o contribución de cualquiera que quiera ofrecer mejoras. Aceptamos [pull requests](#), aunque también puedes rellenar [issues](#) o enviar un correo a daniel-http2@haxx.se con tus sugerencias.

/ Daniel Stenberg

1. Antecedentes

Este es un documento que describe http2 desde un nivel técnico y de protocolo. Comenzó como una presentación, que hice en Estocolmo en abril de 2014, para más tarde extender y convertirse en un documento completo con todo detalle y explicaciones concisas.

RFC 7540 es el nombre oficial de la especificación final de http2 que ha sido publicada el 15 de Mayo de 2015:

<http://www.rfc-editor.org/rfc/rfc7540.txt>

Todos los errores encontrados en este documento son míos propios (y del traducción), resultado de mis propios defectos. Por favor, reportarlos y haré las actualizaciones con sus correcciones.

He intentado utilizar consecuentemente la palabra "http2" para describir el nuevo protocolo, aunque en términos puramente técnicos, el nombre correcto es HTTP/2. He escogido esta opción para favorecer la legibilidad y conseguir un lenguaje más fluido.

Esta es la traducción al español de la versión 1.13 del documento publicada el 12 de septiembre de 2015.

1.1 Autor

Mi nombre es Daniel Stenberg y trabajo en Mozilla. Llevo trabajando con open source y networking durante más de veinte años en numerosos proyectos. Posiblemente se me conozca por ser el desarrollador principal de curl y libcurl. He formado parte del grupo de trabajo HTTPbis durante mucho años, y allí he estado al tanto de las actualizaciones de HTTP 1.1 y me he involucrado en el trabajo de estandarización de http2.

Email: daniel@haxx.se

Twitter: [@bagder](https://twitter.com/bagder)

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2 ¡Ayuda!

Si encuentras errores, omisiones o mentiras descaradas en este documento, por favor envíame un versión actualizada del párrafo afectado y haré versiones modificadas. ¡Se mencionará en los créditos a todo aquel que eche una mano!. Espero ir mejorando este documento a lo largo del tiempo.

El documento está disponible en <http://daniel.haxx.se/http2>

1.3 Licencia



Este documento está licenciado bajo Creative Commons Attribution 4.0 license:

<http://creativecommons.org/licenses/by/4.0/>

1.4 Historial del documento

La primera versión de este documento fue publicada el 25 de abril de 2014. A continuación se muestran las versiones más recientes de este documento:

Versión 1.13

- Convertida la versión maestra a sintaxis Markdown
- 13: Mención a más recursos. Actualización de links y descripciones
- 12: Actualización de la descripción de QUIC y referencia a su draft
- 8.5: Actualizado con números actuales
- 3.4: La media es ahora de 40 conexiones TCP
- 6.4: Actualizada para reflejar lo que dice la especificación

Versión 1.12

- 1.1: HTTP/2 es ahora un RFC oficial
- 6.5.1: enlace al RFC de HPACK
- 9.1: Mención al parámetro de configuración de Firefox 36+ para http2
- 12.1: Añadida sección sobre QUIC

Versión 1.11

- Montón de mejoras en el lenguaje, apuntadas mayormente por contribuciones amigas.
- 8.3.1: mención a actividades específicas de nginx y Apache httpd

Versión 1.10

- 1: El protocolo ha sido “okayed”
- 4.1: Actualizada la palabra, ya que 2014 es el año pasado.
- portada: añadida imagen y nombrado “http2 explicado”, enlace arreglado
- 1.4: añadido el historial del documento
- Corregidos muchos errores de deletreo y gramática
- 14: añadido agradecimiento a reportes de bugs
- 2.4: (mejora) etiquetas para el gráfico de crecimiento HTTP
- 6.3: corregido el orden de los vagones en el tren multiplexado
- 6.5.1: HPACK draft-12

Versión 1.9

- Actualización a HTTP/2 draft-17 y HPACK draft-11
- Añadida la sección “10. http2 en Chromium” (== ahora, una página más larga)
- Montón de correcciones de deletreo
- Ahora en 30 implementaciones
- 8.5: añadidos algunos números de uso actuales
- 8.3: mención también a internet explorer
- 8.3.1 añadido “implementaciones pendientes”
- 8.4.3: mencionar que TLS también eleva el índice de éxito

2. HTTP hoy

HTTP 1.1 se ha convertido en un protocolo usado por prácticamente todo el mundo en Internet. Existen inversiones enormes realizadas en protocolos e infraestructura para aprovecharlo. Esto se ha interpretado cómo que a menudo es más fácil hacer funcionar algo sobre HTTP, que construir algo propiamente nuevo.

2.1 HTTP 1.1 es enorme

Cuando se creó HTTP y fue liberado al mundo, fue concebido como un protocolo más bien simple y sencillo, pero el tiempo ha demostrado lo contrario. HTTP 1.0 en el RFC 1945 es una especificación de 60 páginas publicada en 1996. El RFC 2616 que describe HTTP 1.1, fue publicado sólo tres años más tarde, en 1999 y creció considerablemente hasta las 176 páginas. Todavía, cuando desde el IETF trabajamos en la actualización de la especificación que fue separada en seis documentos, con un número mucho mayor de páginas en total (resultando en el RFC 7230 y familia). De cualquier modo, HTTP 1.1 es grande e incluye una gran variedad de detalles y sutilezas, sin olvidar una gran cantidad de importantes piezas opcionales.

2.2. Un mundo de opciones

La naturaleza de HTTP 1.1 con multitud de pequeños detalles y opciones disponibles para extensiones posteriores, ha generado un ecosistema de software que ha hecho que casi ninguna implementación esté enteramente completada – y realmente es imposible decir que significa implemente por completo la especificación. Esto ha provocado que funcionalidades poco usadas en un principio, no hayan sido comúnmente implementadas ni por supuesto usadas.

Más tarde, esto ha causado un problema de interoperabilidad, cuando clientes y servidores han comenzado a utilizar esas funcionalidades. HTTP Pipelining es el ejemplo principal de este tipo de funcionalidad.

2.3. Uso inadecuado de TCP

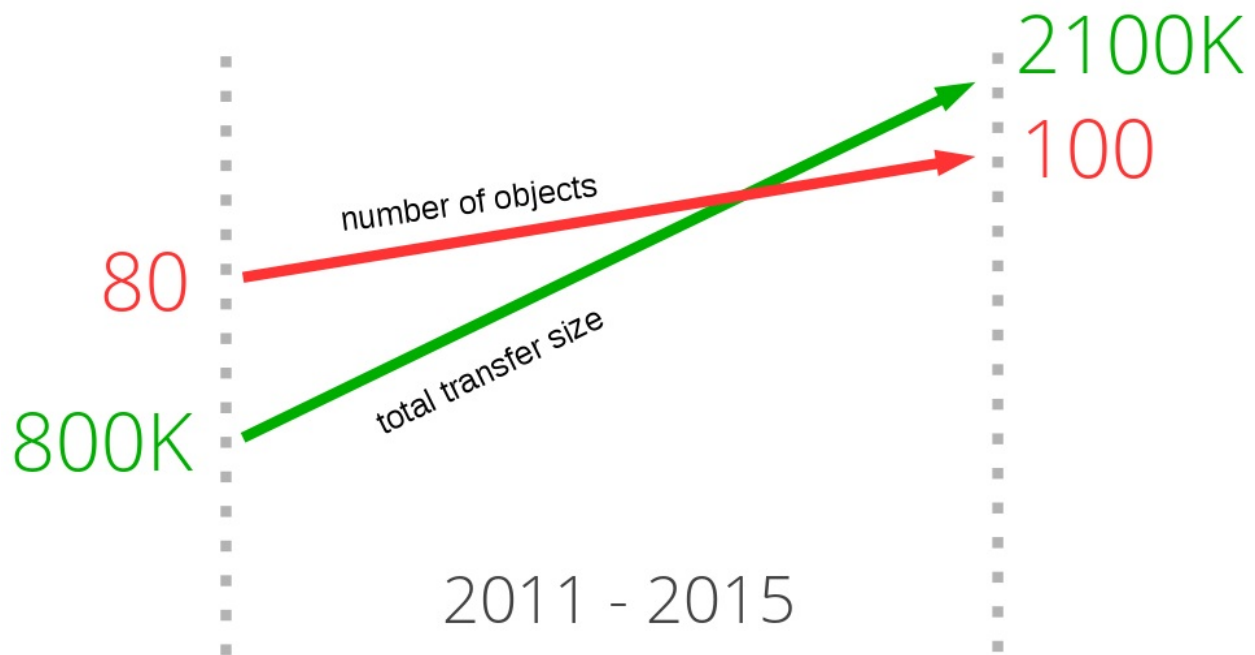
HTTP 1.1 nunca ha conseguido aprovechar la ventajas de todo el potencial y rendimiento que ofrece TCP. Los clientes HTTP y los navegadores tienen que ser muy creativos para encontrar soluciones que reduzcan los tiempos de carga de las páginas.

Han existido otros intentos en paralelo en los últimos años, que han confirmado que no es sencillo reemplazar TCP, y que por lo tanto hay que seguir mejorando tanto TCP, como los protocolos sobre éste. Simplemente, TCP puede usarse mejor evitando pausas o momentos de tiempo que pueden usarse para enviar o recibir más información. Las siguientes secciones vienen a describir algunos de estos defectos.

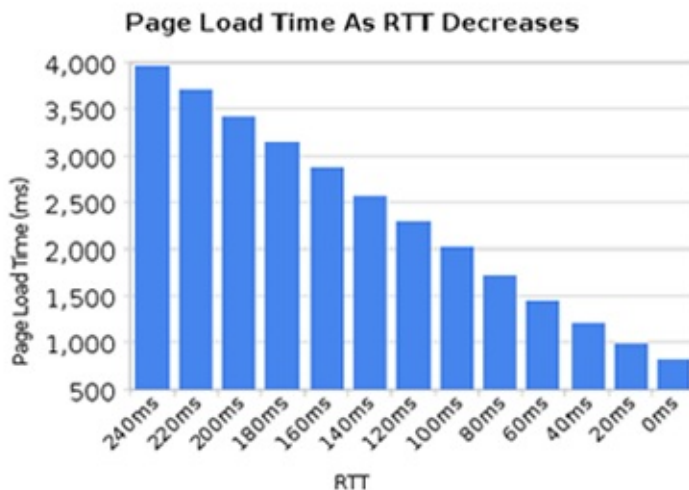
2.4. Tamaños de transferencia y número de recursos

Al observar la tendencia en los sitios web más populares en la carga de su página principal, emerge un patrón muy claro. En los últimos años la cantidad de información que debe ser consumida ha ido elevándose gradualmente hasta más allá de 1.9MB. Lo que es más importante en este contexto, es que de media, se necesitan más de cien recursos individuales para mostrar cada página.

Como se muestra en el siguiente gráfico, la tendencia ha estado en marcha durante un tiempo, y no hay indicación clara de que vaya a cambiar próximamente. Muestra el tamaño total de transferencia (en verde) y el número total de peticiones de media (en rojo) para servir los sitios web más populares del mundo, así como su evolución en los últimos cuatro años.



2.5. Latencia asesina



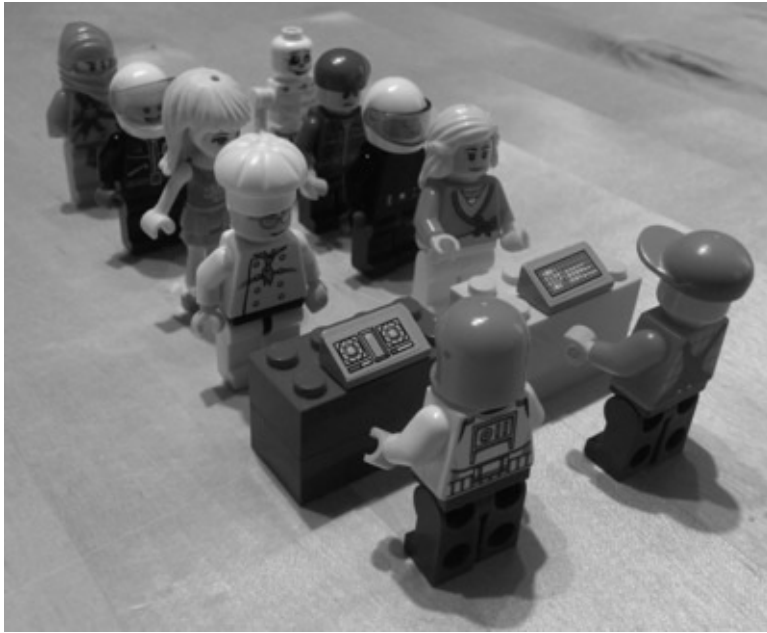
HTTP 1.1 es muy sensible a la latencia, en parte debido a que “HTTP Pipelining” todavía cuenta con demasiados problemas como para seguir apagado para un gran porcentaje de usuarios.

En los últimos años hemos ido viendo como aumentaba el ancho de banda disponible para las personas. No se ha alcanzado el mismo nivel de mejora reduciendo la latencia. Enlaces de alta latencia, como es el caso de las tecnologías móviles actuales, hacen muy complicado conseguir una buena y sobre todo rápida experiencia de usuario en web, incluso contando con un gran ancho de banda.

Otro caso de uso típico que necesita de enlaces con latencia baja, son algunos tipos de vídeo, como vídeo conferencias, juegos u otros casos similares donde no es enviado únicamente un flujo pre-generado de vídeo.

2.6. Bloqueo del primero de la fila

“HTTP Pipelining” es la manera de enviar otra solicitud mientras se está esperando a la respuesta de la solicitud anterior. Es muy similar a esperar en el mostrador de un banco o supermercado. Tú nunca sabes si la persona delante de ti es un cliente rápido, o uno molesto que estará mil horas antes de irse: bloqueo del primero de la fila (“Head of line blocking”).



Por supuesto que puedes tener cuidado a la hora de escoger una cola y escoger la que creas vaya a ir más rápido, incluso a veces podrás iniciar tu propia cola. Pero al final siempre habrá que tomar una decisión, y una vez esté tomada, no podrás cambiar de fila.

Crear una nueva fila supone una penalización en el rendimiento y el uso de recursos, de manera que no es escalable más allá de un número pequeño de filas. No existe una solución perfecta a este problema. Incluso hoy, en 2015, los navegadores son publicados con la opción “HTTP pipelining” deshabilitada por defecto. Se puede encontrar más información sobre esta materia leyendo por ejemplo la [entrada 264354](#) de Firefox bugzilla.

3. Estrategias para evitar los dolores de latencia

Como siempre que se enfrentan distintos problemas, la gente consigue reunir distintas técnicas para solventarlos. Algunas técnicas son inteligentes y muy útiles, otras son trampas horribles.

3.1 Spriting



Spriting es el término que describe la técnica de unir varias imágenes pequeñas, en una única imagen más grande. Más tarde a través de CSS o Javascript, se recortan ciertos pedazos de la imagen grande para ir mostrando las imágenes más pequeñas.

Un sitio web usaría este truco por velocidad. En HTTP 1.1, descargar una única imagen grande es mucho más rápido que descargarse individualmente 100 pequeñas.

Por supuesto que tiene ciertas desventajas en sitios web donde sólo se quieren mostrar uno o dos imágenes pequeñas. De la misma manera, todas las imágenes serán descartadas de la cache al mismo tiempo, en lugar de las imágenes que sean más utilizadas.

3.2 Inlining

Inlining es otro truco para evitar enviar imágenes individuales, cosa que se consigue utilizando URLs “data:” desde un fichero CSS. Tiene beneficios e inconvenientes similares al caso de “spriting”.

```
.icon1 {  
    background: url(data:image/png;base64,<data>) no-repeat;  
}  
  
.icon2 {  
    background: url(data:image/png;base64,<data>) no-repeat;  
}
```

3.3 Concatenación

Un gran sitio web puede tener un montón de ficheros javascript diferentes. Ciertas herramientas de front-end ayudan a los desarrollares a juntar todos ellos en un único paquete gigante de manera que el navegador deberá descargar un único fichero en lugar de docenas de ficheros más pequeños. Se envía mucha información cuando se necesita poca. Un único cambio, fuerza el refresco de mucha información.

La mayoría de las veces, esta técnica es un inconveniente para los desarrolladores implicados.

3.4 Sharding
































El truco de rendimiento final que mencionaré es denominado a menudo fragmentación (“sharding”). Básicamente se basa en servir distintos elementos de tu servicios desde el mayor número de servidores posible. En un primer vistazo puede parecer extraño, pero hay algo razonable detrás de todo ello.

Inicialmente la especificación HTTP 1.1 fijaba que un cliente tenía permitido utilizar un máximo de dos conexiones TCP para cada dominio. Para conseguir no saltarse la especificación, ciertos sitios astutamente se intentaron nuevos nombre de dominio y – voilá – ya disponían de más conexiones de manera que descendían los tiempos de carga.

Con el paso del tiempo, se ha eliminado dicha limitación y los clientes actuales utilizan fácilmente 6-8 conexiones por nombre de dominio, contando con esta limitación, algunos sitios siguen utilizando esta técnica para aumentar el número de conexiones. A medida que el número de objetos ha ido aumentando – como he mostrado anteriormente – utilizar un alto número de conexiones asegura que HTTP rinde bien y hace que tu sitio sea rápido. No es inusual que un sitio web utilice más de 50 o incluso se llegue casi a 100 conexiones en un sitio web utilizando esta técnica. Estadísticas recientes de httparchive.org muestran que las primeras 300.000 URLs del mundo, necesitan una media de 40(!) conexiones TCP para mostrar una página, y la tendencia parece ir incrementando lentamente en el tiempo.

Otra razón es poner las imágenes y otros recursos similares en otro nombre de dominio que no utilice ninguna cookie, ya que el tamaño de las cookies es bastante significativo. Utilizar imágenes sin cookies puede mejorar el rendimiento simplemente consiguiendo ¡peticiones HTTP mucho más pequeñas!

La imagen abajo muestra una traza para una petición de uno de los sitios principales de Suecia, de manera que las peticiones se distribuyen en diferentes nombres de dominio.

● 200	GET		174.jpg	w.cdn-expressen.se	jpeg	6.14 KB	→ 105 ms
● 200	GET		174.jpg	y.cdn-expressen.se	jpeg	4.19 KB	→ 172 ms
● 200				z.cdn-expressen.se	jpeg	4.48 KB	→ 223 ms
● 200				z.cdn-expressen.se	jpeg	4.58 KB	→ 173 ms
● 200				x.cdn-expressen.se	jpeg	35.18 KB	→ 56 ms
● 200				x.cdn-expressen.se	jpeg	12.97 KB	→ 165 ms
● 200				y.cdn-expressen.se	jpeg	4.83 KB	→ 56 ms
● 200				y.cdn-expressen.se	jpeg	9.54 KB	→ 228 ms
● 200				w.cdn-expressen.se	jpeg	182.50 KB	→ 285 ms
● 200				w.cdn-expressen.se	jpeg	5.66 KB	→ 104 ms
● 200				y.cdn-expressen.se	jpeg	12.24 KB	→ 287 ms
● 200				y.cdn-expressen.se	jpeg	6.85 KB	→ 225 ms
● 200				z.cdn-expressen.se	jpeg	7.50 KB	→ 173 ms
● 200				z.cdn-expressen.se	gif	2.85 KB	→ 227 ms
● 200				w.cdn-expressen.se	jpeg	50.87 KB	→ 188 ms
● 200				w.cdn-expressen.se	jpeg	6.65 KB	→ 55 ms
● 200	GET		265.jpg	y.cdn-expressen.se	jpeg	6.09 KB	→ 196 ms
● 200	GET		540.jpg	z.cdn-expressen.se	jpeg	16.14 KB	→ 67 ms
● 200	GET		540.jpg	w.cdn-expressen.se	jpeg	19.89 KB	→ 112 ms
● 200	GET		174.jpg	z.cdn-expressen.se	jpeg	5.03 KB	→ 55 ms
● 200	GET		540.jpg	w.cdn-expressen.se	jpeg	21.27 KB	→ 108 ms
● 200	GET		540.jpg	x.cdn-expressen.se	jpeg	5.43 KB	→ 237 ms
● 200	GET		174.jpg	y.cdn-expressen.se	jpeg	6.08 KB	→ 169 ms
● 200	GET		174.jpg	w.cdn-expressen.se	jpeg	5.62 KB	→ 105 ms
● 200	GET		540.jpg	x.cdn-expressen.se	jpeg	20.32 KB	→ 241 ms
● 200	GET		174.jpg	z.cdn-expressen.se	jpeg	6.66 KB	→ 55 ms
● 200	GET		540.jpg	x.cdn-expressen.se	jpeg	11.13 KB	→ 237 ms
● 200	GET		265.jpg	w.cdn-expressen.se	jpeg	5.20 KB	→ 111 ms
● 200	GET		265.jpg	x.cdn-expressen.se	jpeg	6.93 KB	→ 288 ms
● 200	GET		265.jpg	x.cdn-expressen.se	jpeg	12.09 KB	→ 249 ms
● 200	GET		265.jpg	z.cdn-expressen.se	jpeg	5.92 KB	→ 167 ms
● 200	GET		original.jpg	y.cdn-expressen.se	jpeg	64.28 KB	→ 192 ms
● 200	GET		original.jpg	w.cdn-expressen.se	jpeg	21.88 KB	→ 106 ms
● 200	GET		540.jpg	w.cdn-expressen.se	jpeg	18.77 KB	→ 112 ms
● 200	GET		128.jpg	z.cdn-expressen.se	jpeg	3.34 KB	→ 55 ms
● 200	GET		265.jpg	x.cdn-expressen.se	jpeg	13.00 KB	→ 245 ms
● 200	GET		265.jpg	y.cdn-expressen.se	jpeg	9.19 KB	→ 194 ms
● 200	GET		540.jpg	w.cdn-expressen.se	jpeg	13.13 KB	→ 108 ms
● 200	GET		174.jpg	y.cdn-expressen.se	jpeg	5.66 KB	→ 197 ms
● 200	GET		174.jpg	z.cdn-expressen.se	jpeg	5.56 KB	→ 55 ms
● 200	GET		174.jpg	w.cdn-expressen.se	jpeg	5.07 KB	→ 111 ms
● 200	GET		174.jpg	z.cdn-expressen.se	jpeg	6.16 KB	→ 59 ms
● 200	GET		174.jpg	y.cdn-expressen.se	jpeg	6.57 KB	→ 210 ms
● 200	GET		174.jpg	y.cdn-expressen.se	jpeg	4.58 KB	→ 12 ms
● 200	GET		265.jpg	y.cdn-expressen.se	jpeg	11.49 KB	→ 173 ms

4. Actualizando HTTP

¿No estaría bien hacer un protocolo mejorado? Algo incluyendo...

1. Hacer que el protocolo sea menos sensible a RTT.
2. Arreglar el pipelinig y el problema del primero de la cola.
3. Parar la necesidad y el deseo de seguir aumentando el numero de conexiones para cada host.
4. Mantener todas las interfaces existentes, todo el contenido todos los formatos de URI y sus esquemas.
5. Esto habría de hacerse con el grupo de trabajo del IETF, HTTPbis.

4.1. IETF y el HTTPbis working group

El Internet Engineering Task Force (IETF) es una organización que desarrolla y promueve estándares en Internet. La mayoría a nivel de protocolo. Son ampliamente conocidos por las series de documentos RFC que vienen a documentar todo, desde TCP, DNS, buenas prácticas FTP, HTTP y numerosas variaciones de protocolos que nunca han llegado a ningún sitio.

Los grupos de trabajo dentro IETF son formados con un ámbito limitado a conseguir un objetivo concreto. Establecen un “capítulo” (chapter) con un conjunto de guías y limitaciones de lo que deberían ser. Todo el mundo y cualquiera puede unirse al debate y al desarrollo. Cualquiera que se une y dice algo, tienen el mismo peso y la misma oportunidad de afectar al resultado y todo el mundo es una persona humana e individual, sin importar demasiado en que empresa trabaja el individuo en concreto.

El grupo de trabajo HTTPbis (más tarde se explica este nombre) fue formado durante el verano de 2007 y se le encomendó la tarea de crear una actualización para la especificación HTTP 1.1. Aunque el debajo dentro del grupo comenzó realmente a finales de 2012. El trabajo de de actualización de HTTP 1.1 fue terminado a comienzos de 2014 y resultó en la serie [RFC 7320](#).

La reunión operativa supuestamente final del grupo de trabajo HTTPbis tuvo lugar a comienzos de 2014 en Nueva York. Las debates restantes y los procedimientos del IETF necesarios para publicar el RFC se sucedieron durante el año siguiente.

Algunos de los agentes más grandes en HTTP no estuvieron en los debates o reuniones del grupo de trabajo. No pretendo citar ninguna empresa o producto en particular en este documento, pero claramente algunos actores del Internet de hoy, parecen confiar plenamente en que el IETF hará un buen trabajo sin tener en cuenta a estas empresa.

4.1.1. La parte “bis” del nombre

El grupo se ha denominado HTTPbis de manera que la parte “bis” procede del [adverbio en latín para “dos”](#). Bis es muy usado por el IETF bien como un sufijo o bien como parte de un nombre para una actualización o para la segunda parte de una especificación. En este caso para HTTP 1.1.

4.2. http2 empieza en SPDY

[SPDY](#) es un protocolo encabezado y desarrollado por Google. Ha sido indudablemente desarrollado en abierto y todo el mundo ha sido invitado a participar, pero obviamente son los principales beneficiados al controlar una implementación muy popular de navegador y una parte significativa de la población de servidores con bastantes servicios muy conocidos.

Cuanto el grupo HTTPbis decidió que era hora de comenzar a trabajar en http2, SPDY había demostrado que era un concepto que funcionaba. Había demostrado que era posible desplegar en Internet y habían publicado números demostrando su rendimiento. El trabajo para http2 comenzó posteriormente desde el draft SPDY/3 que se convirtió en el draft-00 de http2 básicamente con un poco de búsqueda y reemplazo.

5. Conceptos de http2

Entonces, ¿qué consigue http2? ¿Donde está el límite para lo que el grupo HTTPbis tiene encargado hacer?

Eran en realidad bastante estrictos y dejaban muy poca oportunidad para la capacidad de innovación dentro del equipo.

- Tiene que mantener los paradigmas HTTP. Se mantiene como un protocolo que envía peticiones al servidor a través de TCP.
- Las URLs `http://` y `https://` no pueden ser cambiadas. No puede existir un nuevo esquema para esto. La cantidad de contenido usando estas URLs es demasiado grande para pretender que cambie.
- Servidores y Clientes HTTP1 se mantendrán durante décadas, deberemos ser capaces de hacer un proxy hacia servidores http2.
- Así pues, los proxies deberán ser capaces de mapear funcionalidades http2 a clientes HTTP 1.1 una a una.
- Eliminar o reducir partes opcionales del protocolo. Esto no era tanto un requisito, como un mantra que llegaba desde SPDY y el equipo de Google. Asegurándose de que todo es obligatorio, no hay manera de que no implementes algo ahora que se convierta en una trampa más adelante.
- No más versiones menores. Se decidió que tanto clientes como servidores serían o no compatibles con http2. Si aparece una necesidad de extender el protocolo o modificar las cosas, entonces habrá nacido http3. No hay más versiones menores en http2.

5.1. http2 para esquemas URI existentes

Como se mencionado con anterioridad, los esquemas de URI existentes, no podrán ser modificados, así que http2 deberá construirse usando los esquemas existentes. Como actualmente se están utilizando en HTTP 1.x, obviamente necesitamos una manera de actualizar el protocolo a http2 o solicitar al servidor de otra manera la utilización de http2 en lugar de protocolos más obsoletos.

HTTP 1.1 tenía una manera definida para hacer esto denominada la cabecera “Upgrade:”, que permitía al servidor enviar una respuesta usando el nuevo protocolo al recibir ¡la a través del protocolo viejo. Todo con el coste de una petición (“round-trip”).

Esa penalización de un round-trip, no era algo aceptable por el equipo de SPDY, y ya que su implementación de SPDY únicamente funcionaba sobre TLS, desarrollaron una nueva extensión TLS utilizada para atajar la negociación de manera significativa. Utilizando esta extensión, denominada NPM (Next Protocol Negotiation), el servidor comunica al cliente que protocolo conoce, así el cliente puede proceder y utilizar su protocolo preferido.

5.2. http2 para `https://`

Se ha prestado mucha atención para conseguir un correcto comportamiento de http2 sobre TLS. SPDY funciona únicamente sobre TLS y se ha intentado con mucha fuerza que TLS sea igualmente obligatorio para http2, pero no se ha conseguido un consenso así que http2 se ha publicado con TLS opcional. Aún así, dos de los fabricantes más destacados han dicho que únicamente implementarán http2 sobre TLS: la iniciativa Mozilla Firefox y Google Chrome. Hoy por hoy, dos de los navegadores principales.

Las razones para escoger solo-con-TLS incluyen el respeto por la privacidad del usuarios y medidas precoces que mostraban un mayor índice de éxito en nuevos protocolos sobre TLS. Esto se debe a la suposición de cualquier tráfico por el puerto 80 es HTTP 1.1, de manera que cualquier dispositivo que intercepte el tráfico puede interferir y destruirlo cuando se trata de otro protocolo distinto a HTTP 1.1.

La obligatoriedad de TLS ha sido causa de mucho movimiento y voces agitadas en las listas de correo y las reuniones – ¿es bueno o es malo? Es un tema infectado – ¡ten esto en cuenta cuando se lo preguntes cara a cara a un miembro de HTTPbis!

De manera similar, ha habido un fiero y largo debate sobre si http2 debe imponer la lista de cifrados que deben ser obligatorios al usar TLS, o sobre si crear una lista negra de unos cuantos o si por el contrario no se debiera obligar nada a la capa TLS, y dejar la decisión al grupo de trabajo TLS. La especificación ha determinado que TLS debe ser al menos la versión 1.2, y ha impuesto algunas restricciones en el cifrado.

5.3. Negociación http2 sobre TLS

Next Protocol Negotiation (NPN), es el protocolo utilizado para negociar SPDY con servidores TLS. Como no se trataba de un estándar adecuado, fue llevado a la IETF y de aquello surgió ALPN: Application Layer Protocol Negotiation. ALPN es lo que se está promoviendo para ser usado con http2, mientras que los clientes y servidor SPDY seguirán utilizando NPN.

El hecho de que NPN estuviera primero y que ALPN haya tardado un poco en el proceso de estandarización, ha llevado a que muchos clientes y servidores http2 hayan implementado de forma prematura soporte para ambas extensiones para la negociación de http2. Igualmente como NPN se utiliza para SPDY y muchos servidores ofrecen tanto SPDY como http2, dar soporte tanto para NPN como ALPN tiene perfecto sentido.

La principal diferencia de ALPN respecto a NPN está en quién decide que protocolo hablar. Con ALPN el cliente le indica al servidor el listado de protocolos en el orden de preferencia, y el servidor escoge el que él quiere, mientras que con NPN es el cliente quien toma esa última decisión.

5.4. http2 for http://

Como se ha mencionado brevemente con anterioridad, para HTTP 1.1 en texto plano, la manera de negociar http2 es solicitar al servidor mediante una cabecera Upgrade:. Si el servidor habla http2, responderá con un estado “101 Switching” y a partir de entonces hablar http2 en esa conexión. Por supuesto que te das cuenta que este procedimiento de actualización está costando un viaje de red “round-trip”, pero por otra parte la conexión http2 debería ser mantenida y reutilizada de manera más generalizada que las conexiones HTTP1.

Aunque algunos representantes de navegadores han declarado que no implementarán este modo de hablar http2, el equipo de Internet Explorer ha comunicado que ellos lo harán, así como curl, que ya soporta actualmente esta modo.

6. El protocolo http2

Suficiente sobre antecedentes, historia y asuntos políticos que nos han traído hasta aquí. A continuación vamos a bucear en asuntos específicos del protocolo. Los bits y los conceptos que crean http2.

6.1. Binario

http2 es un protocolo binario.

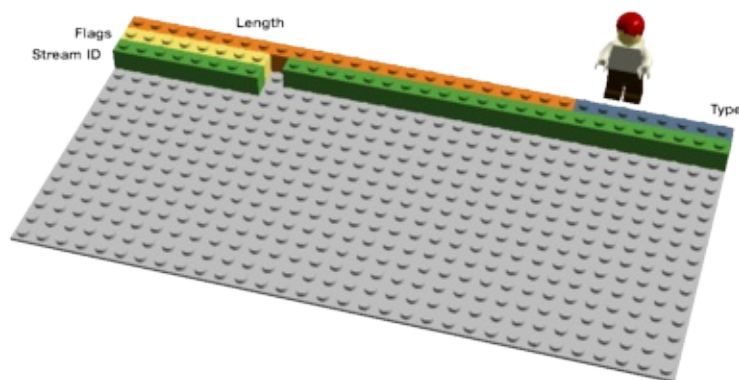
Déjame asentar esto un minuto. Si eres una persona que ha estado involucrada en protocolos de Internet, las posibilidades de que reacciones instintivamente contra esta opción y de que calcules argumentos que expliquen como los protocolos de texto/ascii son superiores por permitir a los humanos hacer consultas a mano con telnet. http2 es en binario para conseguir hacer el entramado (framing) mucho más sencillo. Determinar el comienzo y el final de cada trama es una de las cosas realmente complicadas en HTTP 1.1 así como en todos los protocolos en texto en general. Eliminando espacios en blanco opcionales y distintas formas de escribir la misma cosa, las implementaciones serán mucho más simples.

Igualmente hace mucho más fácil separar las partes de protocolo de las tramas en si, lo que en HTTP1 está confusamente entremezclado.

El hecho de las funcionalidades de compresión del protocolo y de que casi siempre correrá sobre TLS, restan importancia a que el protocolo no sea en texto plano, ya que la información tampoco viajaría en texto de cualquier manera. Simplemente tendremos que acostumbrarnos a utilizar el inspector de Wireshark o algo similar para determinar que está pasando a nivel de protocolo con http2.

El Debugging en este protocolo deberá hacerse utilizando herramientas como curl o analizando el tráfico de red con el disector de tráfico http2 de Wireshark u otra herramienta similar.

6.2. El formato binario



http2 envía tramas en binario. Pueden enviarse distintos tipos de trama, y todos ellos comienzan de la misma manera:

Tipo, Tamaño, Flags, Identificador de Flujo y la carga útil de la trama.

Existen 10 tipos de tramas definidos en la especificación http2 y los dos tipos fundamentales que se mapean con las funcionalidades de HTTP 1.1 son DATA y HEADERS. Más adelante en el documento, voy a describir algunas de las tramas en algo más de detalle.

6.3. Flujos multiplexados

El identificador de flujo mencionado en la sección anterior en la descripción de la trama binaria, asocia cada trama enviada a través de http2 con un “flujo”. Un flujo es una asociación lógica. Una secuencia de tramas independiente bidireccional intercambiados entre el cliente y el servidor dentro de una conexión http2.

Una conexión http2 puede contener múltiples flujos abiertos concurrentes, ya sea con tramas de finalización de distintos flujos. Los flujos pueden ser establecidos y usados unilateralmente por el cliente o el servidor, y pueden ser cerrados por cualquiera de los dos puntos. El orden en el que se envía cada flujo es significativo. Los receptores procesan las tramas en el orden en el que son recibidos.

La multiplexación de los flujos significa que paquetes de distintos flujos se mezclan en la misma conexión. Dos (o más) trenes independientes se convierten en uno único y luego son separados en el otro punto. Aquí están los dos trenes:



Los dos trenes multiplexados sobre la misma conexión:



6.4. Prioridades y dependencias

Cada flujo tiene una prioridad (denominada también "peso"), que se usa para indicar al peer que flujos deben ser considerados más importantes en caso de contar con restricciones de recursos que obliguen al servidor a seleccionar que flujo enviar primero.

Utilizando la trama PRIORITY, un cliente puede indicar al servidor que otro flujo, es dependiente un flujo. Esto permite al cliente construir un "árbol" de prioridades, en el que varios "flujos hijos", pueden depender de que varios "flujos padre" sean completados.

Los pesos de prioridad y las dependencias pueden ser cambiados dinámicamente en tiempo real, lo que permitirá en una página con muchas imágenes por ejemplo, que el navegador cambie la prioridad en las solicitudes a medida que el usuario hace scroll; o al cambiar entre tabs, puede priorizar el conjunto de flujos que acaba de coger el foco del usuario.

6.5. Compresión de Cabeceras

HTTP es un protocolo sin estado. De manera resumida significa que cada petición debe indicar al servidor los detalles necesarios para que la petición sea atendida, sin que el servidor tenga que almacenar gran cantidad de información y meta-información de peticiones anteriores. Ya que http2 no cambia para nada este paradigma, debe cumplirlo igualmente.

Esto convierte el HTTP en repetitivo. Cuando un cliente solicita muchos recursos de un mismo servidor, como imágenes para una página web, habrán una gran serie de solicitudes que parecerán prácticamente idénticas. Una serie de algo casi idéntico, está pidiendo compresión a gritos.

Mientras el número de objetos sigue en aumento, como ya se ha comentado con anterioridad, el uso y el tamaño de las cookies ha seguido creciendo igualmente durante este tiempo. Estas cookies deben ser incluidas en todas las peticiones. Mayormente las mismas cookies en cada petición.

El tamaño en las peticiones HTTP 1.1 ha ido haciéndose tan grande en los últimos tiempos que han terminado siendo más grandes que la ventana TCP inicial, lo que hace que el envío de la petición sea muy lento al necesitar recibir un ACK de vuelta del servidor antes de completar la petición por completo. Este sería otro argumento más para la compresión.

6.5.1. La compresión tiene truco

Las compresiones HTTPS y SPDY han resultado ser vulnerables a ataques [BREACH](#) y [CRIME](#). Insertando texto conocido en el flujo, y averiguando que es lo que cambia en el resultado comprimido, un atacante puede averiguar que se está enviando.

Hacer compresión en contenido dinámico para un protocolo evitando las vulnerabilidades de estos dos ataques, requiere reflexiones y consideraciones cuidadosas. Esto es lo que el grupo HTTPbis intenta hacer.

Se introduce [HPACK](#), Header Compression for HTTP/2, que – como sugiere adecuadamente su nombre – es un formato de compresión especialmente diseñado para cabeceras http2 y estrictamente hablando, está siendo especificado en otro borrador separado de Internet. El nuevo formato, junto con otras contra-medidas como un bit que solicita a los intermediarios no comprimir una cabecera específica o el desplazamiento opcional de tramas deberían hacer mucho más difícil llegar a romper esta compresión.

En palabras de Roberto Peon (uno de los creadores de HPACK):

“HPACK ha sido diseñado para dificultar que una implementación conforme filtre información, para hacer que la codificación y decodificación muy rápida y barata, para proporcionar al receptor control del tamaño del contexto de la compresión, para permitir que un proxy reindice (por ejemplo, estados compartidos entre un frontal y una trasera a través de un proxy), y para una comparación rápida de cadenas codificadas mediante huffman”.

6.6. Reset - piensa diferente

Uno de los inconveniente con HTTP es que cuando un mensaje HTTP ha sido enviado con cierto tamaño especificado en la cabecera Content-Length, no puedes ser parado fácilmente. A menudo (aunque no siempre), se puede cerrar la conexión TCP, pero pagando el precio de tener que negociar un handshake TCP de nuevo.

Una solución mejor para esto es simplemente parar el mensaje, y comenzar uno nuevo. En http2, esto puede hacerse con la trama RST_STREAM, que previene derrochar el ancho de banda y permite mantener la conexión.

6.7. Server push

Esta funcionalidad también se conoce como “cache push”. La idea aquí es que si el cliente solicita un recurso X determinado, el servidor determina que es altamente probable que el cliente solicite también el recurso Z, de manera que es enviado sin que el cliente lo solicite. Esto ayudará al cliente a tenerlo en su cache, de manera que ya estará allí cuando sea necesario.

El envío desde el servidor (Server push) es algo que un cliente debe permitir explícitamente, e incluso si está permitido, podrá a su propia elección rápidamente terminar el flujo con un RST_STREAM no permitiendo así ningún Server Push en particular.

6.8. Flow Control (Control de Flujo)

Cada flujo individual sobre http2 tiene su propia ventana de flujo anunciada, sobre la que el otro extremo puede enviar información. Si conoces como funciona SSH, es muy similar en estilo y espíritu.

Para cada flujo, cada uno de los extremos tiene que indicar a su peer que tiene más espacio para la información entrante, de manera que el otro extremo únicamente puede enviar tanta información como espacio se dispnga, hasta que la ventana sea hecha más grande. Únicamente existe control de flujo para las tramas DATA.

7. Extensiones

El protocolo obliga que a un receptor a leer e ignorar todas las tramas desconocidas utilizando tipo de trama desconocida (unknown frame types). Dos partes pueden así negociar el uso de nuevos tipos de trama en una base hop-by-hop, y las tramas no podrán cambiar el estado y no tendrán control de flujo.

Si http2 debía permitir o no extensiones, fue un tema ampliamente debatido durante el tiempo de desarrollo del protocolo con opiniones igualmente balanceadas a favor y en contra. Después del draft-12, el péndulo basculó por última vez, y las extensiones se permitieron de nuevo.

Por lo tanto, las extensiones no son partes del protocolo real por lo que son documentadas al margen de la especificación del núcleo. Llegados a este punto, existen dos tipos de trama que se debatieron para su inclusión en el protocolo, y que probablemente serán las primeras tramas en ser extensiones. Las describiré a continuación por su popularidad y por su estado anterior como tramas “nativas”:

7.1. Alternative Services (Servicios Alternativos)

Durante la adopción de http2, existen razones para sospechar que las conexiones TCP serán mucho más largas y se mantendrán vivas mucho más tiempo de lo que han estado las conexiones con HTTP 1.x. Un cliente debe ser capaz de hacer un montón de las cosas que hace con una única conexión hacia cada dominio/sitio, así que una conexión estará abierta por un tiempo bastante elevado.

Esto afectará a como funcionan los balanceadores HTTP y se generarán situaciones en las cuales un sitio querrá anunciar y sugerir que el cliente se conecte a otro host, tanto por razones de rendimiento, como porque el sitio puede necesitar mantenimiento u otra razón similar.

El servidor enviará entonces la cabecera [Alt-Svc: header](#) (o la trama ALTSVC en http2) que indicará al cliente el servidor alternativo. Otra ruta al mismo contenido, utilizando otro servicio, dominio y número de puerto.

El cliente intentará conectarse a ese servicio asíncronamente, y utilizará ese servicio alternativo si éste funciona correctamente.

7.1.1. TLS oportunista

La cabecera Alt-Svc permite al servidor que proporciona el contenido por http://, informar al cliente que ese mismo contenido está también disponible a través de una conexión TLS.

Esta es una funcionalidad debatida en cierta manera. Una conexión de ese tipo, realizaría una conexión TLS sin autenticar que no sería advertida como “segura” en ningún sitio, ya que no usaría un candado en la Interfaz de Usuario o avisaría al usuario de cualquier manera que no se trata del viejo HTTP plano, sino de que se trata de TLS oportunista, concepto que en sí mismo, que encuentra firmes detractores.

7.2. Blocked (Bloqueado)

Una trama de este tipo está pensada para ser enviada únicamente una vez por un agente http2 cuando aún queda información que enviar, pero el control de flujo lo prohíbe. La idea de esto es que si tu implementación recibe esta trama, entonces sabes que tu implementación ha cometido algún error y/o están recibiendo menos que la velocidad que puedes recibir por esta razón.

Se cita en el draft-12, antes de que esta trama fuera expulsada como extensión:

“La trama BLOCKED está incluida en esta revisión para facilitar la experimentación. Si los resultados no son positivos, se eliminará”

8. Un mundo http2

Entonces, ¿Cómo serán las cosas cuando http2 sea adoptado? ¿Será adoptado?

8.1. Cómo afectará http2 en humanos normales

http2 aún no está ampliamente ni desplegado ni usado. No podemos decir como serán las cosas. Hemos observado como se ha usado SPDY y se pueden hacer suposiciones y cálculos basados en éste y otros experimentos pasados y presentes.

http2 reduce el número de “round-trips” de red necesarios y con la multiplexación y el descarte rápido de flujos no deseados, evita el dilema del bloqueo del primero de la fila.

El protocolo permite un alto número de flujos paralelos más allá de cualquier sitio que actualmente utilice la técnica de dominios fragmentados (sharding).

Utilizando la funcionalidad de prioridades correctamente en los flujos, hay altas probabilidades de que el cliente realmente reciban la información importante antes de la información menos priorizada. Juntando todo esto, diría que existe una muy buena oportunidad de que se esté apuntando a tiempos de carga mucho más rápidos así como hacia sitios web más reactivos. En pocas palabras: un mejor experiencia web.

No creo que se pueda decir todavía cuánto más rápido o cuantas mejoras vamos a llegar a ver. Para empezar la tecnología está todavía en un fase muy temprana y aún no se ha comenzado a ver clientes y servidores ajustando implementaciones para aprovechar todo el poder que nos está ofreciendo el nuevo protocolo.

8.2. Cómo afectará http2 al desarrollo web

Durante los años los desarrolladores web así como sus entornos de desarrollo, han ido recopilando una gran colección de trucos y herramientas para solventar los problemas con HTTP 1.1, de los que he hablado al comienzo del este documento como justificación principal para http2.

Muchos de estos atajos que tanto herramientas como desarrolladores utilizan por defecto sin pensar, probablemente afectarán negativamente el rendimiento de http2, o al menos hará que no se aprovechen los nuevos súper poderes de http2. Tanto “spriting” como “inlining”, probablemente no deban utilizarse con http2. La fragmentación en varios dominios (“sharding”), será perjudicial en http2, ya que probablemente haya más beneficio utilizando menos conexiones..

El principal problema es por supuesto que tanto los sitios web como los desarrolladores web, necesitarán desarrollar y desplegarse en un mundo en el que a corto plazo al menos, existan clientes tanto HTTP1.1 como http2, de manera que el reto será conseguir el máximo rendimiento sin tener que ofrecer dos frontales diferentes.

Únicamente por estas razones, sospecho que pasará algo de tiempo antes de que veamos alcanzado el máximo potencial de http2.

8.3. Implementaciones http2

Intentar documentar las implementaciones específicas en un documento como este, es por supuesto hacerlo en vano y está condenado al fracaso porque estará desfasado en un periodo muy corto de tiempo. En lugar de esto, explicaré la situación en un término más amplio y simplemente haré una referencia para los lectores hacia a la [lista del implementaciones](#) en el sitio web de http2.

Han existido una gran cantidad de implementaciones desde un primer momento, y este número ha ido incrementándose durante el trabajo con http2. Mientras se escribe esto, existen más de 30 implementaciones listadas, la mayoría implementando la versión final.

Firefox ha sido el navegador que ha estado encabezando los borradores más nuevos. Twitter ha estado a la altura ofreciendo sus servicios sobre http2. Google ha comenzado a ofrecer soporte http2 en algunos servidores de pruebas desde abril de 2014 y desde mayo de 2014, han ofrecido soporte http2 en las versiones de desarrollo de Chrome. Microsoft ha presentado que soporta http2 desde una “tech preview” de su próxima versión de Internet Explorer.

Tanto curl como libcurl soportan http2 inseguro así como basado en TLS utilizando una de las distintas bibliotecas TLS.

[H2O](#), [Apache Traffic Server](#) y [nghttp2](#) han publicado todos ellos servidores con soporte para http2.

8.3.1. Implementaciones pendientes

Las dos opciones más populares en servidores web, Apache HTTPD y Nginx ofrecer soporte para SPDY, pero todavía no han publicado soporte oficial para http2 en ningún release. Nginx ha publicado un “[alpha patch](#)” así como el módulo de Apache para HTTP/2 denominado [mod_h2](#) parece que está encaminado a ser incluido en una release pública muy pronto.

8.4. Críticas comunes a http2

Durante el desarrollo de este protocolo ha existido cierto debate en determinados aspectos, y algunas personas creen que el protocolo se ha diseñado completamente mal. Me gustaría mencionar algunas de las quejas más comunes, así como los argumentos en su contra:

8.4.1. “El protocolo está diseñado o hecho por Google”

Existen variaciones que implican un mundo todavía más dependiente o controlado por Google. No es cierto. El protocolo ha sido desarrollado desde el IETF de la misma manera en la que se han venido desarrollando protocolos en los últimos 30 años. De cualquier manera, todo reconocemos el impresionante trabajo hecho por Google con SPDY que no solo demostró que era posible desplegar un nuevo protocolo sino que también aportó los números que indicaban que se podrían conseguir mejoras.

Google ha [anunciado](#) públicamente¹ que van a retirar el soporte para SPDY y NPN en Chrome a partir de 2016, y que aconsejan a los servidores utilizar HTTP/2 en su lugar.

8.4.2. “El protocolo solo es útil para navegadores”

Esto es medio verdad. Una de las principales razones para el desarrollo de http2, es arreglar HTTP pipelining. Si tu caso de uso no tiene problema, entonces es probable que http2 no te suponga demasiada mejora. Aunque ésta no sea la mejora principal en el protocolo, es bastante significativa.

Según distintos servicios se vayan dando cuenta del gran poder y posibilidades que tienen los flujos multiplexados a través de una única conexión, sospecho que veremos más aplicaciones utilizando http2.

Pequeñas APIs REST o uso más simples de programación con HTTP 1.x puede que no encuentren que el salto a http2 ofrezca demasiadas ventajas. Aunque igualmente, tampoco deberían existir demasiados inconvenientes con http2 en la mayoría de los casos.

8.4.3. “El protocolo solo es útil para sitios grandes”

Para nada. La capacidad de multiplexar mejorará notablemente la experiencia de usuario para sitios pequeños con conexiones de altas latencias sin distribuciones geográficas. Los sitios más grandes suelen ser más rápidos, ofreciendo conexiones distribuidas, con tiempos “round-trip” más cortos para los usuarios.

8.4.4. “Su uso de TLS lo hace más lento”

Esto puede ser cierto hasta cierto punto. El “handshake” TLS añade un poco de tiempo extra, pero actualmente se está trabajando en reducir el número de viajes “round-trips” necesarios para TLS. El trabajo extra para hacer TLS a través de la conexión comparado con texto plano no es insignificante y es claramente notable al usar más CPU y energía para un mismo patrón de tráfico sobre protocolo no seguro. Cuantificar cuanto y su impacto está sujeto a opiniones y medidas. El sitio istlsfastyet.com ofrece por ejemplo una fuente de información.

Telecom y otros operadores de red, por ejemplo la ATIS Open Web Alliance, han dicho que necesitan tráfico no cifrado¹ para ofertar cacheo, compresión u otras técnicas necesarias para ofertar una experiencia de usuario rápida a través de satélites, en aviones por ejemplo.

http2 no establece el uso de TLS como obligatorio, así que no se deberían combinar los términos. Muchos usuarios de Internet han expresado preferencia en un uso más extenso de TLS para que podamos ayudar a proteger la privacidad de los usuarios.

Ciertos experimentos han demostrado que usar TLS aporta un mayor índice de éxito que implementar nuevos protocolos en texto plano a través del puerto 80 ya que existen demasiados dispositivos desplegados en el mundo que pueden interferir con aquello que crean que es HTTP1.1 si es que va por el puerto 80 y puede parecer HTTP a veces.

Por último, gracias a la multiplexación de flujos de http2 en una única conexión, en el caso de uso de navegadores normales, se realizarán muchos menos “handshakes” TLS, así que se conseguirá un rendimiento más rápido que utilizando HTTPS sobre HTTP 1.1.

8.4.5. “No ser ASCII es un factor decisivo”

Si, nos gusta ser capaces de ver los protocolos claramente, ya que lo hace mucho más fácil para debuggear y tracear. Pero los protocolos basados en texto son más susceptibles a error, y provocan muchísimos errores en su interpretación.

Si no soportas un protocolo binario, entonces estas descartando TLS o la compresión en HTTP 1.x, que han estado utilizando muchísimo tiempo.

8.4.6. “No es más rápido que HTTP/1.1”

Por supuesto que es algo sujeto a debate y a discusión en cómo se mide que significa más rápido, pero ya en los días de SPDY, se realizaron multitud de pruebas que demostraban que la página cargaba más rápido (por ejemplo “[How Speedy is SPDY?](#)” de la gente de la Univesidad de Washington y “[Evaluating the Performance of SPDY-enabled Web Servers](#)” por Hervé Servy) y dichos experimentos se han repetido también con http2. Tengo ganas de ver publicados los resultados de esas pruebas y experimentos.

Una primera prueba básica realizada por httpwatch.com podría implicar que HTTP/2 cumple sus promesas.

8.4.7. “No respeta las capas”

En serio, ¿es eso tu argumento? Las capas no son pilares intocables de una religión global y si hemos pasado algunas zonas grises al hacer http2, ha sido en interés de hacer un protocolo bueno y efectivo a partir de las premisas iniciales.

8.4.8. “No arregla varios defectos de HTTP/1.1”

Eso es cierto. Con el objetivo específico de mantener los paradigmas HTTP/1.1, hemos tenido que mantener ciertas funcionalidades anticuadas. Por ejemplo ciertas cabeceras comunes como las temidas cookies, cabeceras de autenticación y más. Como contrapunto al mantenimiento de estos paradigmas, se ha conseguido un protocolo que puede desplegarse sin la obligación de sustituir o reescribir una inconcebible cantidad de trabajo en dicha actualización. http2 es básicamente una nueva capa de “framing”.

8.5. ¿Estará http2 desplegado masivamente?

Es demasiado pronto para afirmarlo con seguridad, pero puedo suponer y hacer una estimación, y eso es lo que haré aquí.

Los “no-nos” dirán “mira que bien hecho está IPv6” como ejemplo de un nuevo protocolo que ha necesitado décadas para empezar a estar ampliamente desplegado. Aunque http2 no es IPv6. Es un protocolo por encima de TCP que usa los mecanismos de actualización normales de HTTP, sus números de puerto, TLS, etc. No va a necesitar que cambien para nada la mayoría de routers o firewalls.

Google demostró al mundo con su trabajo con SPDY que un nuevo protocolo como este puede ser desplegado y utilizado por navegadores y servicios desde distintas implementaciones en un periodo de tiempo razonablemente corto. Aunque la cantidad de servidores en Internet ofreciendo SPDY está en ñle rango del 1%, la cantidad de información que manejan esos servidores es mucho más grande. Algunos de los sitios web más populares actualmente en el mundo, ofrecen SPDY.

http2, basado en los mismos paradigmas básicos que SPDY, diría que tiene más probabilidades de ser desplegado desde que es un protocolo del IETF. El despliegue de SPDY siempre fue algo retenido por el estigma de “ser un protocolo de Google”.

Hay varios grandes navegadores detrás del despliegue. Representantes de Firefox, Chrome e Internet Explorer han expresado que ofrecerán navegadores con soporte http2 y ya han mostrado implementaciones funcionando.

Existen grandes operadores de servidor que van a ofrecer http2 pronto, entre los que se incluye Google, Twitter y Facebook así como esperamos ver soporte http2 pronto en servidores web populares como Apache HTTP Server y nginx. H2o es un nuevo increíblemente rápido servidor HTTP con soporte http2que tiene potencial.

Algunos de los fabricantes de proxy más grandes, incluyendo HAProxy, Squid and Varnish han manifestado intenciones de añadir soporte http2.

Casi a finales de 2015, la cantidad de tráfico en http2 ha continuado incrementándose. A comienzos de septiembre, el uso en Firefox 40 era del 13% del total del tráfico HTTP y el 27% del tráfico HTTPS, mientras que Google está recibiendo aproximadamente un 18% de trafico HTTP/2. Hay que tener en cuenta que Google está experimentando con nuevos protocolos (ver QUIC en la sección 12.1), lo que hace bajar las medidas de uso para http2.

9. http2 en Firefox

Firefox ha estado siguiendo la pista a los borradores muy de cerca, y ofreciendo implementaciones http2 de prueba durante muchos meses. Durante el desarrollo del protocolo http2, los clientes y servidores tienen que ponerse de acuerdo sobre qué versión del borrador están utilizando, lo cual dificulta levemente ejecutar pruebas. Hay que asegurarse de que el cliente y el servidor implementan la misma versión del borrador del protocolo.

9.1. Primero, asegurar que está activado

Desde la versión 35, publicada el 13 de enero de 2015, Firefox soporta http2 por defecto.

Entrar en 'about:config' en la barra de direcciones, y buscar la opción denominada "network.http.spdy.enabled.http2draft". Habrá que asegurarse que está puesta a true. Firefox 36 añadió otra opción de configuración llamada "network.http.spdy.enabled.http2" que está a true por defecto. Ésta última controla la versión simple de http2, mientras que la primera activa y desactiva la negociación de las revisiones de los borradores de http2. Ambas opciones están activadas desde Firefox 36.

9.2. Sólo TLS

Recordad que Firefox únicamente soporta http sobre TLS. Solo verás http2 en acción con Firefox al entrar en sitios con https:// que ofrezcan soporte http2.

9.3. ¡Transparente!

The screenshot shows the Firefox Network Inspector interface. The 'Network' tab is active, displaying a list of requests. The selected request is a GET request to 'https://twitter.com/'. The 'Headers' pane on the right shows the response headers. The 'X-Firefox-Spdy: h2-12' header is highlighted with a red box, indicating that HTTP/2 is being used. Other visible headers include 'Cache-Control: no-cache, no-store, max-age=0', 'Content-Encoding: deflate', 'Content-Type: text/html; charset=utf-8', 'Date: Wed, 07 May 2014 08:49:40 GMT', 'Expires: Tue, 31 Mar 1981 05:00:00 GMT', 'Last-Modified: Wed, 07 May 2014 08:49:40 GMT', 'Pragma: no-cache', and 'Server: tfe'.

No existe ningún elemento en la interfaz de usuario que nos diga que se está hablando http2. No es fácilmente detectable. Un modo de averiguarlo es activar "Web developer->Network" y comprobar las cabeceras de respuesta para ver que está enviando el servidor. La respuesta será entonces "HTTP/2.0" y algo más. Firefox inserta su propia cabecera denominada "X-Firefox-Spdy:" como se aprecia en la captura de pantalla de arriba. Las cabeceras que se ven en la herramienta de red al utilizar http2 han sido convertidas desde el formato binario de http2, para parecerse al estilo clásico de las cabeceras de HTTP 1.x. No existe ningún elemento en la interfaz de usuario que nos diga que se está hablando

http2. No es fácilmente detectable. Un modo de averiguarlo es activar “Web developer->Network” y comprobar las cabeceras de respuesta para ver que está enviando el servidor. La respuesta será entonces “HTTP/2.0” y algo más. Firefox inserta su propia cabecera denominada “X-Firefox-Spdy.” como se aprecia en la captura de pantalla de arriba.

Las cabeceras que se ven en la herramienta de red al utilizar http2 han sido convertidas desde el formato binario de http2, para parecerse al estilo clásico de las cabeceras de HTTP 1.x.

9.4. Visualizar el uso de http2

Existen plugins de Firefox disponibles que ayudan a visualizar si un sitio está usando HTTP/2. Uno de ellos es [“SPDY Indicator”](#).

10. http2 en Chromium

El equipo de Chromium ha implementado y soportado http2 en sus canales dev y beta desde hace bastante tiempo. Desde Chrome 40, publicado el 27 de enero de 2015, http2 está activado por defecto para algunos usuarios. El número de estos, comenzó muy pequeño, y ha ido aumentando a lo largo del tiempo.

El soporte para SPDY será cancelado próximamente. En una entrada de blog, el proyecto anunció lo siguiente en [febrero de 2015](#):

“Chrome ha soportado SPDY desde Chrome 6, pero al tener la mayor parte de las ventajas presentes en HTTP/2, es hora de decir adiós. Hemos planificado quitar el soporte para SPDY a comienzos de 2016”

10.1. primero, asegurar que está activado

Introducir “chrome://flags/#enable-spdy4” en la barra de direcciones del navegador y hacer click en “activar” (“enable”), si no está previamente activado.

10.2. Sólo TLS

Recordar que Chrome sólo implementa http2 sobre TLS. Únicamente se verá http2 en acción con Chrome, al visitar sitios con https:// que ofrezcan soportes http2.

10.3. Visualizar el uso de http2

Existen plugins de Chrome disponibles que ayudan a visualizar si un sitio está usando HTTP/2. Uno de ellos es “[SPDY Indicator](#)”.

10.4. QUIC

Los experimentos actuales de Chrome con QUIC (ver sección 12.1), diluyen de alguna manera los número de HTTP/2.

11. http2 en curl

El [proyecto curl](#) ha estado ofreciendo soporte experimental para http2 desde septiembre de 2013.

Siguiendo el espíritu de curl, pretendemos ofrecer todos los aspectos de http2 en la medida de nuestras posibilidades. Es un uso común de curl se usado como una herramienta de testeo, y una manera de “pingar” manualmente sitios web, y es nuestra intención mantenerlo igualmente para http2.

curl utiliza una librería externa [nghttp2](#), para implementar la funcionalidad de la capa trama http2. curl necesita la versión nghttp2 1.0 o superior.

Actualmente en Linux, curl y libcurl no están siempre desplegados con el soporte activado para HTTP/2.

11.1. Parecido a HTTP 1.x

Internamente curl convertirá las cabeceras http2 al estilo de cabeceras HTTP 1.x, y serán entregadas al usuario, de manera que aparecerán de maneras similar al HTTP tradicional. Esto facilitará la transición a todos los usos actuales de curl. De manera similar, curl convertirá las cabeceras salientes con el mismo estilo. Serán pasadas en estilo HTTP 1.x y las convertirá al vuelo cuando se esté hablando con servidores http2. Esto permitirá que los usuarios no se preocupen demasiado sobre la versión HTTP en particular que esté siendo utilizada.

11.2. Texto plano, inseguro

curl soporta http sobre TCP estándar utilizando la cabecera “Upgrade:”. Si realizas una petición HTTP solicitando http2, curl preguntará al servidor si es posible actualizar la conexión a http2.

11.3. Qué bibliotecas TLS

curl tiene soporte para una amplia variedad de bibliotecas TLS distintas para su implementación TLS, y esto sigue siendo válido para el soporte http2. El desafío de TLS en el mundo http2, es el soporte ALPN y en cierta medida el soporte NPN.

Compila curl con versiones modernas de OpenSSL o NSS, para obtener soporte ALPN y NPN. Utiliza GnuTLS o PolarSSL y soportará ALPN pero no NPN.

11.4. Uso de línea de comando

Para indicar a curl que utilice http2, tanto e texto plano como sobre TLS, hay que utilizar la opción `--http2` (Esto es “menos menos http2”). De momento curl por defecto ofrece HTTP/1.1 así que es necesario una opción extra cuando se quiere http2.

11.5. Opciones de libcurl

11.5.1 Activar HTTP/2

Tu aplicación utilizará URLs normales con `https://` o `http://`, pero habrá que indicar el parámetro `CURLOPT_HTTP_VERSION` de `curl_easy_setopt` a `CURL_HTTP_VERSION_2` para intentar que libcurl utilice http2. Intentará de la mejor forma conectar con http2, pero volverá a HTTP 1.1 si éste falla.

11.5.2 Multiplexación

libcurl intenta mantener comportamientos existentes, por lo que se hace necesario que se active la multiplexación HTTP/2 en tu aplicación mediante la opción [CURLMOPT_PIPELINING](#). De lo contrario se seguirá utilizando una petición en cada momento por conexión.

Otro pequeño detalle a tener en cuenta es que al solicitar varias transferencias al mismo tiempo con libcurl, usando su interfaz multiple, una aplicación puede empezar varias transferencias al mismo tiempo, y que si se pretende que libcurl espere e introduzca todas ellas por la misma conexión en lugar de abrir nuevas conexiones, existe la opción [CURLOPT_PIPEWAIT](#).

11.5.3 Server push

A partir de la versión libcurl 7.44.0 se da soporte para la funcionalidad HTTP/2 server push. Puedes utilizarla indicando un callback con la opción [CURLMOPT_PUSHFUNCTION](#).

Si la aplicación acepta el push, se creará una nueva transferencia con un manejador fácil CURL, y se enviará contenido, como cualquier otra transferencia.

12. Después de http2

Se han tomado muchas decisiones duras y compromisos en http2. Con el despliegue de http2 por delante, se ha determinado una forma para actualizar a otras versiones, que sienta las bases para tener nuevas revisiones del protocolo más adelante. Se ha introducido el concepto y la infraestructura necesarias para manejar múltiples versiones en paralelo. ¿Quizás no es necesario discontinuar completamente lo viejo para introducir algo nuevo?

http2 lleva al futuro un montón de “legado” HTTP 1 con la intención de mantener posible tráfico de proxy de ida y vuelta entre HTTP 1 y http2. Parte de ese legado dificulta aún más el desarrollo y la inventiva. ¿Quizás http3 se deshará de parte de este legado?

¿Qué crees que falta todavía en http?

12.1. QUIC

El protocolo de Google, [QUIC](#) (Quick UDP Internet Connections) es un experimento muy interesante, desarrollado con el mismo estilo y espíritu con el que en su día se hizo SPDY. QUIC es un sustituto para TCP + TLS + SPDY implementado usando UDP.

QUIC permite la creación de conexiones con mucha menos latencia, soluciona la pérdida de paquetes al sólo bloquear flujos individuales en lugar de todos a la vez como hacer HTTP/2, y posibilita la creación de conexiones por distintas interfaces de red fácilmente – así también cubre otras áreas que MPTCP pretende resolver.

QUIC de momento está únicamente implementado por Google en Chrome y en sus servidores, y es código no fácilmente reutilizable en otras partes, aunque [libquic](#) es un esfuerzo intentado conseguir eso exactamente. La especificación es todavía algo vaga y cambia rápidamente. Ya existe un [borrador](#) en el grupo de trabajo de transporte del IETF.

13. Otras lecturas

Si encuentras este documento un poco ligero de contenido o de detalles técnicos, aquí tienes otros recursos para ayudar a satisfacer tu curiosidad:

- La lista de correo HTTPbis y su archivo: <http://lists.w3.org/Archives/Public/ietf-http-wg/>
- La especificación http2 actual en su versión en HTML: <https://httpwg.github.io/specs/rfc7540.html>
- Detalles de networking http2 de Firefox: <https://wiki.mozilla.org/Networking/http2>
- Detalles de la implementación http2 de curl: <http://curl.haxx.se/dev/readme-http2.html>
- El sitio web de http2: <http://http2.github.io/> y quizás su FAQ en particular: <http://http2.github.io/faq/>
 - El capítulo sobre HTTP/2 de Ilya Grigorik's en su libro "High Performance Browser Networking": <http://chimera.labs.oreilly.com/books/1230000000545/ch12.html>

14. Agradecimientos

Inspiración y la imagen del paquete con formato Lego de Mark Nottingham.

Los datos de tendencias HTTP vienen de <http://httparchive.org>.

El gráfico RTT viene de las presentaciones hechas por Mike Belshe.

A mis hijos Agnes y Rex por prestarme sus figuritas Lego para la imagen de “head of line”.

Gracias a los siguientes amigos por las revisiones y el feedback: Kjell Ericson, Bjorn Reese, Linus Swälas and Anthony Bryan. Se aprecia mucho vuestra ayuda que ¡ha mejorado de verdad este documento!

Durante varias iteraciones, las siguientes personas han reportado bugs y sugerido mejoras al documento: Mikael Olsson, Remi Gacogne, Benjamin Kircher, saivlis, florin-andrei-tp, Brett Anthoine, Nick Parlante, Matthew King, Nicolas Peels, Jon Forrest, sbrickey, Marcin Olak, Gary Rowe, Ben Frain, Mats Linander, Raul Siles, Alex Lee, Richard Moore

El traductor a español, Javier Infante, quisiera agradecer a Gorka Gorrotxategi por la revisión y sus correcciones sobre este texto.