UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

REDES DE COMPUTADORES I – ELO 322

Proyecto de Investigación

"HTTP/2.0"

Primer Semestre 2016

Profesor Agustín González

Integrantes Millenka Jachura
Joaquín Rodríguez
Sebastián Yuste

RESUMEN

HTTP/2.0 es una mejora de uno de los protocolos conocidos, por lo que se abordaran sus características y funcionamiento. Para esto es necesario primer conocer el contexto en el cual se desarrolló este protocolo, por lo que se iniciara hablando brevemente sobre la historia subyacente a este, seguido de sus principales características, para luego investigar más a fondo la estructura de este, mencionando las fallas encontradas en HTTP/1.1, la diferencias que este posee con su predecesor, y las innovaciones que este último propuso para mejorar el rendimiento, tales como el tipo de cifrado utilizado, y más particularmente, un cambio significativo a la hora del intercambio de información, presentando las ideas de flujo, multiplexaciónn, frames, server-push entre otros. Se dará a conocer el estado actual de la implementación de HTTP/2.0. Y finalmente se demostrara con un ejemplo práctico la superioridad de HTTP/2.0 frente a HTTP/1.1 mediante la descarga de una página web y posterior análisis de los paquetes recibidos con ambos protocolos.

INTRODUCCIÓN

Para poder navegar por Internet como lo hacemos hoy en día, ciertamente se han requerido años de desarrollo y con ello un protocolo capaz de manejar los requerimientos de cada sitio de la web. Fue así como nació HTTP, un protocolo de la capa de aplicación creado con el fin de gestionar la descarga de las distintas páginas que visitamos.

Fue en 1991 cuando se lanza por primera vez HTTP/0.9, un protocolo muy básico y simple, esto ya que no incluía cabeceras y solo contaba con el comando GET, por lo tanto, el protocolo funcionaba casi en su totalidad de bajada del servidor, era muy poca la información que el cliente podía comunicar al servidor.

Ya 5 años después, en 1996, éste se actualiza a su versión /1.0, la cual significó un gran avance gracias a la inclusión de cabeceras y nuevos comandos: POST y HEAD. Ahora el cliente si podía tener más participación en la comunicación, podía entregar y postear información a la web.

La versión por excelencia HTTP/1.1 no llegaría hasta 1999. Con solo saber que es la más utilizada hasta hoy en día podemos dar cuenta de su significancia. Esta versión trajo consigo nuevos comandos, tales como PUT y DELETE, involucrando más aún al cliente. Pero su incorporación más significante fueron las conexiones persistentes (con y sin pipeline), ahora no era necesario generar más de una conexión TCP para descargar el contenido de la web, revolucionando en su momento la rapidez de navegación.

En 2009 Google lanzó un protocolo experimental llamado SPDY, el objetivo principal de éste era solucionar los problemas generados por la mala utilización de HTTP/1.1 (múltiples conexiones no-persistentes para bajar una página web) y por el crecimiento del tamaño de las distintas webs con el paso del tiempo (mayor cantidad de elementos gráficos).

Fue el último protocolo mencionado, SPDY, quien sentó las bases para un avance en el protocolo HTTP. En este proyecto de investigación estudiaremos HTTP/2.0, mostraremos sus principales características y su potencial en la navegación web del futuro, además compararemos éste con el funcionamiento de su antecesor HTTP/1.1, incluyendo así una demostración práctica al final del proyecto.

EL PROTOCOLO DEL FUTURO

Como sus antecesores, HTTP/2.0 es un protocolo de la capa de aplicación, dedicado esencialmente a la navegación web. Este protocolo funciona sobre la capa de transporte, donde TCP es el encargado de conectar al cliente y al servidor, asegurando así también que los paquetes lleguen a su destino. RFC 7540 es la publicación donde está estandarizado HTTP/2.0, desarrollado por Hypertext Transfer Protocol working group del IETF (Internet Engineering Task Force).

Entre sus características más básicas destaca el no ser un protocolo de texto plano como sus antecesores, ahora es directamente binario, para evitar problemas con mayúsculas, minúsculas o espacios en blanco. También con el fin de ser un protocolo seguro, aun cuando esto puede significar disminuir su velocidad, se construyó con la idea de un cifrado punto-punto, es decir, al momento de realizar una conexión, solamente el cliente y el servidor tienes las llaves para des encriptar el mensaje, el tipo de encriptación más utilizado es el mismo proveniente de HTTPS: TLS.

ADIÓS CONEXIONES PARALELAS

Como bien es sabido HTTP/1.1 había incluido en su construcción las conexiones persistentes, las cuales permitían hacer varios requerimientos a través de una única conexión TCP. Pero, con el paso del tiempo, los navegadores de Internet dejaron de lado este tipo de conexiones con el afán de cargar aún más rápido los contenidos. Así comenzaron a utilizar varias conexiones no-persistentes en paralelo, lo cual significó reducir los tiempos de carga, pero malgastaba recursos y ancho de banda al tener que realizar una mayor cantidad de conexiones.

Para solucionar la problemática presentada anteriormente HTTP/2.0 incorporó dos medidas principales, basándose en el SPDY de Google: Una única conexión TCP y el concepto de flujo (stream).

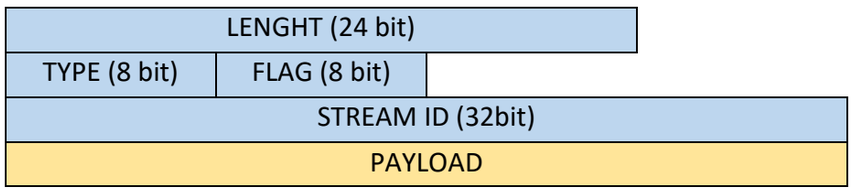
En primer lugar, el concepto de flujo es utilizado como una seguidilla de paquetes llamados frames (más adelante los estudiaremos mejor) que son parte de una misma cadena, enviada a través de la misma conexión TCP. Para hacer este proceso más optimo, se usa la idea de la multiplexación, que consiste en enviar distintos flujos a través de la misma conexión. Estos frames son luego identificados y ordenados por el receptor mediante la información contenida en sus encabezados.

En cuanto a la cantidad de conexiones, los principales problemas que traía el hecho de crear múltiples conexiones eran dos, el inicio de una conexión TCP es un proceso lento de varios pasos. Además, cada una de estas conexiones comienzan a transmitir paquetes con un slow start, lo que provoca que se desperdicie una cantidad importante de tiempo tanteando la congestión existente en la red. De esta forma, al tener solamente una conexión, solo se usa una vez el tiempo de establecimiento de esta, y se puede hacer que transmita la información a la mayor velocidad posible.

MENOS ES MÁS

Uno de los principales problemas con HTTP/1.1 es que cada mensaje de solicitud hacia un mismo servidor contiene información repetida, redundante, el nombre del servidor, el tipo de protocolo, lenguaje, entre otras informaciones que pueden contener los requerimientos HTTP. ¿Es esto necesario? Pues no, en HTTP/2.0 se elimina la información redundante con tal de enviar una menor cantidad de datos y así disminuir las tasas de envío, a esto se le conoce como compresión de cabecera.

Esto es posible gracias a la incorporación del concepto de frame. Un frame es la estructura de mensaje utilizada por HTTP/2.0, compuesta por una cabecera de 9 Bytes y el "*payload*" o datos útiles. Su estructura se muestra a continuación:



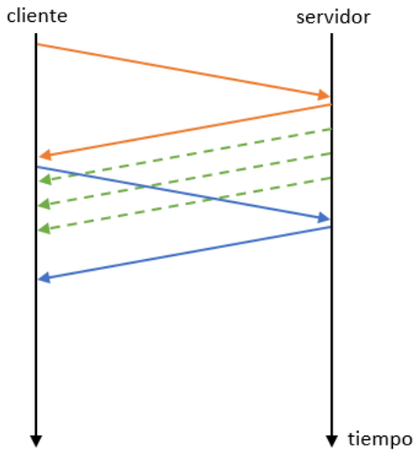
Existen varios tipos de frames: PUSH_PROMISE, DATA, WINDOW_UPDATE, SETTING, entre otros, pero profundizaremos con los frames de tipo HEADER.

Como se mencionó con anterioridad, este nuevo protocolo busca eliminar la información redundante de las cabeceras y para ello, en un flujo de frames, se requiere solo uno de tipo header con toda la información de la conexión al servidor web (tal como realiza HTTP/1.1), luego los frames correspondientes al mismo flujo se reconocerán como una cadena y, por lo tanto, tendrán asignada la información del header en el buffer.

UN PASO MÁS ADELANTE QUE EL CLIENTE

Una de las novedades más revolucionarias que incorpora HTTP/2.0 es el Server-Push, herramienta ejecutada por los servidores para incrementar de manera radical la velocidad de respuesta. Esta herramienta consiste en que el servidor se antepone a los requerimientos del cliente, dicho de otra forma, el servidor se adelanta y prevee cuales contenidos requerirá necesariamente el cliente y comienza enviando pequeños paquetes con dichos contenidos al caché de éste, aun cuando el cliente mismo no ha hecho ningún requerimiento. De esta manera, luego al momento de requerir los contenidos de la página web estos ya estarán en el caché del navegador y, de esta manera, se reducirán de manera considerable los tiempos de respuesta.

En el siguiente diagrama temporal se puede ver un ejemplo del funcionamiento de Server-Push:



En primer lugar, se establece la conexión (color naranjo) y en seguida el servidor envía pequeños paquetes "necesarios" (color verde). El cliente luego realiza un requerimiento, el cual es servidor responde, pero ya parte de ese requerimiento se encontraba en el caché (color azul).

IMPLEMENTACION

Actualmente HTTP/2.0, al ser un protocolo reciente, está en proceso de implementación, es decir, solo algunas páginas ya pueden funcionar sobre HTTP/2.0. Por lo tanto, surge la pregunta ¿Cómo sabe un cliente cuando puede usar medio y cuando no? La solución está en que cada vez que se solicita cierta página, se hace mediante un mensaje de HTTP/1.1, el cual entre la información de su cabecera puede contener un campo llamado Upgrade, que solicita al servidor de la página web un cambio del protocolo mediante el cual se comunican. En caso de que el servidor tenga soporte de este protocolo y quiera realizar el cambio, envía un mensaje de confirmación y comienza la comunicación con este nuevo protocolo, en este caso HTTP/2.0.

Debido a que es decisión de cada sitio web implementar en sus servidores soporte HTTP/2.0, este proceso puede extenderse mucho tiempo. Un ejemplo claro de ello es que del top 10 millones de sitios web más visitados sólo el 7,1% de estos cuentan con soporte HTTP/2.0, aun cuando los principales navegadores como Mozilla Firefox, Google Chrome, Safari, Internet Explorer, entre otros ya tienen el soporte al nuevo protocolo implementado.

DEMOSTRACIÓN PRÁCTICA

Para nuestra demostración nos ayudamos de la prueba de HTTP/2.0 facilitada por Akamai en su sitio web (https://http2.akamai.com/demo). Nuestro método utilizado fue realizar los mismos requerimientos por separado, en primer lugar, con HTTP/1.1 y luego con HTTP/2.0, mientras capturábamos los paquetes a través de Wireshark.

HTTP/1.1 (2,18 s)

HTTP/2.0 (1,74 s)

CAPTURA CON HTTP/1.1 :

383	6.801645	23.56.214.130	192.168.1.36	HTTP	1414	HTTP/1.1 200 OK (PNG)
392	6.801991	192.168.1.36	23.56.214.130	HTTP	414	GET /demo/tile-88.png HTTP/1.1
394	6.802702	192.168.1.36	23.56.214.130	HTTP	414	GET /demo/tile-90.png HTTP/1.1
395	6.803673	23.56.214.130	192.168.1.36	HTTP	1449	HTTP/1.1 200 OK (PNG)
396	6.803874	192.168.1.36	23.56.214.130	HTTP	414	GET /demo/tile-91.png HTTP/1.1
399	6.810235	23.56.214.130	192.168.1.36	HTTP	1358	HTTP/1.1 200 OK (PNG)
401	6.811007	192.168.1.36	23.56.214.130	HTTP	414	GET /demo/tile-92.png HTTP/1.1
404	6.813134	23.56.214.130	192.168.1.36	HTTP	1116	HTTP/1.1 200 OK (PNG)

```
Hypertext Transfer Protocol
  GET /demo/tile-88.png HTTP/1.1\r\n
  Host: http1.akamai.com\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0\r\n
  Accept: */*\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Referer: https://http1.akamai.com/demo/h2_demo_frame.html\r\n
  Connection: keep-alive\r\n
  \r\n
```

```
Hypertext Transfer Protocol
  GET /demo/tile-91.png HTTP/1.1\r\n
  Host: http1.akamai.com\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0\r\n
  Accept: */*\r\n
  Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Referer: https://http1.akamai.com/demo/h2_demo_frame.html\r\n
  Connection: keep-alive\r\n
  \r\n
```

Como se puede observar a simple vista, en ambos paquetes se repite la misma información, excepto por las líneas encerradas en rojo (ya que los requerimientos cambian). Podemos confirmar de manera practica que la redundancia en las cabeceras HTTP/1.1 es una realidad, mal utilizando datos y recursos de red.

CAPTURA HTTP/2.0 :

```
1487 15.495594   23.14.70.157   192.168.1.36   HTTP2   399 HEADERS
1494 15.498084   23.14.70.157   192.168.1.36   HTTP2  1102 DATA, DATA, DATA
1508 15.504667   23.14.70.157   192.168.1.36   HTTP2  1494 DATA, DATA, DATA, DATA
1510 15.506146   23.14.70.157   192.168.1.36   HTTP2  1494 DATA
1515 15.506152   23.14.70.157   192.168.1.36   HTTP2   300 DATA, DATA
1527 15.508670   23.14.70.157   192.168.1.36   HTTP2  1424 DATA, DATA, DATA, DATA, DATA
1534 15.636729   23.14.70.157   192.168.1.36   HTTP2  1102 DATA, DATA, DATA
1541 15.642266   23.14.70.157   192.168.1.36   HTTP2   878 DATA, DATA, DATA
```

```
HyperText Transfer Protocol 2
  Stream: HEADERS, Stream ID: 737, Length 307
    Length: 307
    Type: HEADERS (1)
    Flags: 0x04
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 0000 0000 0000 0000 0010 1110 0001 = Stream Identifier: 737
    [Pad Length: 0]
    Header Block Fragment: 188210011f278586b19272ff1f13a1fe5d0bedb920c0524a...
    [Header Length: 623]
    [Header Count: 16]
    Header: :status: 200
    Header: server: Apache
    Header: etag: "71956da0ecf1574a9352ba09c984a0a8:1461344460"
    Header: last-modified: Fri, 22 Apr 2016 17:01:00 GMT
    Header: accept-ranges: bytes
    Header: content-length: 370070
    Header: content-type: image/x-icon
    Header: cache-control: max-age=2592000
    Header: expires: Sun, 31 Jul 2016 02:57:23 GMT
    Header: date: Fri, 01 Jul 2016 02:57:23 GMT
    Header: access-control-max-age: 86400
    Header: access-control-allow-credentials: false
    Header: access-control-allow-headers: *
    Header: access-control-allow-methods: GET,HEAD,POST
    Header: access-control-allow-origin: *
    Header: strict-transport-security: max-age=31536000 ; includeSubDomains
    Padding: <MISSING>
HyperText Transfer Protocol 2
  Stream: DATA, Stream ID: 737, Length 1518
    Length: 1518
    Type: DATA (0)
    Flags: 0x00
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 0000 0000 0000 0000 0010 1110 0001 = Stream Identifier: 737
    [Pad Length: 0]
    Data: 0000000000000000000000000000000000000000000000000000...
    Padding: <MISSING>
  Stream: DATA, Stream ID: 737, Length 4072
    Length: 4072
    Type: DATA (0)
    Flags: 0x00
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 0000 0000 0000 0000 0010 1110 0001 = Stream Identifier: 737
    [Pad Length: 0]
    Data: 0000000000000000000000000000000000000000000000000000...
    Padding: <MISSING>
  Stream: DATA, Stream ID: 737, Length 4072
  Stream: DATA, Stream ID: 737, Length 4072
```

En esta captura podemos notar la estructura de un frame HTTP/2.0 y el funcionamiento del flujo a grandes rasgos, en primer lugar, un header frame perteneciente al flujo 737 que contiene muchos datos similares a los contenidos en un header HTTP/1.1 y luego otro paquete que contiene 4 data frames correspondientes al mismo flujo 737, como se explicó con anterioridad, estos frames no necesitan información adicional, ya que fue encapsulada por el encabezado header con anterioridad y como pertenecen al mismo flujo en el buffer se asimila el encabezado como si fuese de cada frame.

Sobre el tiempo de carga de cada una de las imágenes, podemos señalar que la nueva versión de HTTP fue sustancialmente más rápida que su antecesora, reduciendo en un 20% el tiempo de carga de la misma imagen.

CONCLUSIÓN

Como hemos visto a lo largo de esta investigación, HTTP/2.0 es un protocolo totalmente innovador con respecto a sus antecesores, tanto Server-Push como la multiplexación de flujos influyen directamente en la rapidez de transmisión del protocolo, son capaces de aprovechar de mejor forma el ancho de banda (al no tener que iniciar más de una conexión y que el servidor inicie su transmisión previamente a los requerimientos). Además, la utilización de los frames y la compresión de cabeceras, permite disminuir la cantidad de información a enviar, eliminando datos redundantes y con ello aprovechar de mejor manera los recursos de la conexión.

Debemos enfatizar en que HTTP/2.0 es el protocolo del futuro, dado que su implementación en la web mundial actual, dada su complejidad e inmensidad, será un proceso lento, pero a medida que este protocolo se masifique se podrán notar realmente los cambios en la rapidez de navegación, hasta acercarse a las experimentales (cerca de un 50% más rápida que con HTTP/1.1 .

REFERENCIAS

https://elbauldelprogramador.com/como-funciona-http2-protocolo-que-acelera-considerablemente-la-navegacion-web/

http://blog.scottlogic.com/2014/11/07/http-2-a-quick-look.html

http://www.hoyandroid.com/http2-asi-va-a-mejorar-la-velocidad-de-tu-navegacion-sin-que-tu-tengas-que-hacer-nada/