UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Estudio comparativo de las versiones más recientes de HTTP

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Tomás Silvestre, Enrique

Tutor: Oliver Gil, José Salvador

2020/2021

Resumen

En este trabajo estudiaremos las características que introducen las últimas versiones del Protocolo de Transferencia de Hipertexto (HTTP) para mejorar las prestaciones de su predecesor, HTTP/1.1.

En primer lugar analizaremos las ventajas que introducen HTTP/2 y HTTP/3, así como los desafíos con los que cuenta cada versión y veremos el nivel de despliegue actual de ambas tecnologías en Internet. A continuación, repasaremos el intercambio de mensajes para poder contemplar las mejoras que introducen estas dos últimas versiones. Por último, plantearemos tres escenarios, para analizar cómo se comporta cada una de las versiones ante distintas simulaciones en la configuración del enlace y para evaluar el comportamiento real en sitios web que hacen uso de ambas versiones.

Palabras clave: HTTP, QUIC, estudio, análisis, rendimiento.

Abstract

In this work we will study the features introduced by the latest versions of the Hypertext Transfer Protocol (HTTP) in order to improve the performance of its predecessor, HTTP/1.1.

First, we will analyze the advantages that HTTP/2 and HTTP/3 introduce, as well as the challenges that each version has to face, and we will see the current level of deployment of both technologies on the Internet. Next, we will review the exchange of messages to be able to contemplate the improvements that these last two versions introduce. Finally, we will propose three scenarios, to analyze how each of the versions behaves in the face of different simulations in the link configuration and to evaluate the real behavior on websites that use both versions.

Keywords : HTTP, QUIC, research, analysis, performance.

Índice de contenidos

Índice de figuras

Índice de gráficos

1. Introducción

El servicio web es, posiblemente, el más popular en Internet desde sus inicios. El protocolo de transferencia de hipertexto, abreviado HTTP, es un protocolo de la capa de aplicación que rige el proceso de comunicación en la web entre el navegador (cliente) y el servidor. Durante más de 20 años se ha venido empleando mayoritariamente la versión HTTP/1.1 del protocolo (una revisión del original 1.0 de 1996), pero en los últimos años, la IETF y el consorcio W3C han propuesto dos nuevas versiones cuyo principal objetivo es mejorar las prestaciones de este protocolo en el contexto de las redes de computadores actuales. Para ello, la versión HTTP/2 propone el uso de codificación binaria de los mensajes en lugar de en modo texto, añade compresión en las cabeceras de los mensajes, emplea tecnologías PUSH para que los servidores anticipen el envío de objetos y mejora la multiplexación de los flujos. Sin embargo, esta segunda versión sigue funcionando sobre el protocolo TCP lo que implica limitaciones a nivel de transporte a las que la versión HTTP/3 intenta sobreponerse empleando el protocolo QUIC sobre UDP. En este trabajo se propone el estudio de estas nuevas versiones, analizando su nivel de despliegue actual y evaluando el grado de mejora alcanzado por cada una de ellas empleando distintos escenarios.

1.1 Objetivos

El objetivo principal que buscamos con la realización de este trabajo es estudiar las características que introducen las versiones de HTTP/2 y HTTP/3 y analizar como estas mejoran su comportamiento respecto a la versión original del protocolo, HTTP/1.1. Del mismo modo, analizar lo que supone el uso de QUIC para el protocolo HTTP y las ventajas que este ofrece.

También se pretende estudiar el funcionamiento del protocolo de la capa de seguridad TLS, debido al gran impacto y la alta relación que tiene con las últimas versiones del protocolo HTTP, además de la importancia que adquiere el papel de la seguridad en las comunicaciones actualmente.

Por otro lado, vamos a estudiar el rendimiento que ofrecen las distintas versiones del protocolo HTTP para los distintos escenarios que plantearemos. Una vez realizadas las pruebas de rendimiento, podremos analizar los resultados y realizar la comparación del comportamiento para las distintas versiones.

1.2 Metodología

La metodología que hemos seguido para la realización del trabajo se divide en dos bloques fundamentales:

En primer lugar, realizamos un estudio en profundidad del protocolo HTTP/2, ya que este era el sucesor de HTTP/1.1 e introduciría un gran número de características nuevas con el objetivo de mejorar las comunicaciones en la red. A continuación, realizamos el mismo estudio sobre HTTP/3, el cual adquiere bastantes características de HTTP/2 y mejoraba otras tantas. Por último, y como consecuencia del gran impacto que tiene sobre estos protocolos y la importancia de la seguridad en Internet, realizamos el estudio del protocolo TLS.

En segunda instancia, después de haber estudiado distintas posibilidades para las pruebas de rendimiento, decidimos instalar servidores propios y configurar una red privada virtual para poder crear nuestros propios escenarios de simulación con distintas configuraciones. En la máquina del servidor instalamos el servidor Apache y lo configuramos para que pudiera trabajar con HTTP/1.1 y HTTP/2, además de la funcionalidad de server push que nos ofrece esta versión.

1.3 Estructura del documento

El presente documento se divide en cinco partes, en las que abordaremos distintos aspectos para el estudio y elaboración de este trabajo.

La primera parte la hemos dedicado a la introducción del trabajo que abordamos, así como para presentar algunos aspectos como los objetivos que buscábamos con la realización del mismo y la metodología seguida para lograr dichos objetivos

En la segunda parte, presentaremos los principales problemas con los que se encontraba el protocolo HTTP/1.1 e introduciremos las principales características que se han propuesto tanto en HTTP/2 como en HTTP/3 para mejorar las prestaciones del protocolo HTTP. Además mostraremos el despliegue actual para cada una de estas tecnologías.

En tercer lugar, después de haber presentado las características que incluye cada versión, analizaremos el intercambio de mensajes para las tres versiones, donde revisaremos las conexiones que se realizan tanto en TCP como en UDP. Además, revisaremos el funcionamiento del protocolo de seguridad en la capa de transporte, TLS, pues adquiere un papel fundamental en el objetivo principal que se busca con las nuevas versiones del protocolo HTTP y se centra en las mejoras para la reducción de la latencia.

Una vez hayamos finalizado el análisis teórico-comparativo de las distintas versiones del protocolo, nos centraremos en la realización de pruebas de rendimiento para analizar el comportamiento de las distintas versiones en la práctica, tanto para un escenario simulado, como para sitios web que hacen uso de las nuevas versiones del protocolo en la actualidad.

Por último, recogeremos las principales conclusiones que hemos extraído tras haber realizado el estudio comparativo con la realización del trabajo, así como sobre los resultados obtenidos para los distintos escenarios prácticos que planteamos. Además, presentaremos algunas propuestas de investigación que se pudieran desarrollar en un futuro, relacionadas con el estudio que hemos realizado.

2. Últimas versiones de HTTP

Desde la primera versión del protocolo HTTP de 1991, pasando por la versión HTTP/1.0 de 1996 y HTTP/1.1 de 1997 hasta prácticamente el 2015 con el lanzamiento de HTTP/2, lo que tenían en común todas todas esas versiones es que tanto las peticiones que realiza el cliente, como las respuestas proporcionadas por el servidor, se realizaban por medio de texto plano (ASCII). Este método de comunicación entre ambas partes resulta muy ineficiente de procesar por parte de los sistemas informáticos, lo que se traduce en altas latencias y retardos. Por otro lado, en las cabeceras nos encontrábamos con campos redundantes, duplicados o con información innecesaria que afectan negativamente al tamaño de los mensajes.

Por defecto, HTTP estaba limitado a procesar las peticiones de manera secuencial. La siguiente petición es realizada una vez que la respuesta a la petición actual ha sido recibida. La descarga secuencial de los recursos implicaba que las descargas de objetos grandes, bloquearan todas las descargas posteriores, lo que obligaba a los clientes a utilizar varias conexiones TCP de manera simultánea para ser capaces de procesar varias solicitudes al mismo tiempo.

La introducción de una arquitectura de pipelining con HTTP/1.1, traía mejoras notables a la hora de establecer las conexiones respecto a las comunicaciones persistentes y no persistentes, aunque este mecanismo todavía tiene algunas limitaciones como el bloqueo de línea de cabecera o la sobrecarga del sistema, sobretodo a nivel de servidor, o las limitaciones en cuanto al número máximo de conexiones permitidas por cliente, establecidas de este modo para evitar posibles ataques DoS.

Figura 1. Evolución del protocolo HTTP.

2.1 SPDY

La versión del protocolo HTTP/1.1 [1], ha sido el principal mecanismo de transmisión de información en Internet desde 1999. En aquellos tiempos, era difícil estimar el impacto y el crecimiento que podía tener Internet y era inimaginable tener en mente el volumen y tipo de información que se maneja hoy en día con las páginas web actuales, por lo que el protocolo no fue diseñado especialmente para tener una baja latencia. Los diseños y el contenido de las elaboradas páginas web, la transmisión de vídeo a gran definición o las transmisiones en vivo requerían que el protocolo experimentara un gran

cambio que pudiera soportar las exigencias de uso y así, poder mejorar la experiencia del usuario.

Con el objetivo de conseguir una mayor eficacia en las comunicaciones por Internet, Google anunció a mediados de 2009 que estaba trabajando en un nuevo protocolo, SPDY [2], que ayudaría a reducir las latencias en las páginas web. SPDY, fue definido por sus desarrolladores como un protocolo de la capa de aplicación para el transporte de contenidos a través de Internet, diseñado específicamente para conseguir una latencia mínima.

El protocolo HTTP mantiene sus bases intactas a pesar de la incorporación de SPDY, mismos métodos, cabeceras y semántica. Según el modelo OSI, el protocolo SPDY opera en la capa de sesión, sobre TCP. La capa de sesión es la encargada de la gestión de las conexiones entre dos extremos, por tanto, SPDY no reemplaza al protocolo HTTP, sino que lo complementa, modifica algunas funcionalidades del protocolo original y añade otras distintas que permiten mejorar las comunicaciones con la finalidad de reducir la latencia.

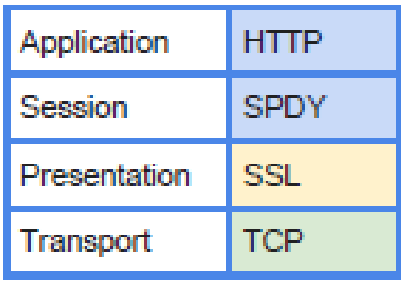

Figura 2. Pila de protocolos con SPDY según el modelo OSI. Chromium

Entre las características que incorporaba la propuesta de Google encontramos la transmisión de datos en formato binario, la introducción de flujos multiplexados, la priorización de peticiones, la compresión de las cabeceras, los flujos iniciados por el servidor y el uso obligatorio de cifrado de datos para las comunicaciones. Todas estas características se desarrollarán en el siguiente apartado.

Aplicando todas estas nuevas funcionalidades en su protocolo, Google consiguió mejoras de hasta el 60% en los tiempos de carga de las webs que utilizaban el protocolo SPDY.

2.2 HTTP/2

A principios del año 2012, basándose en el proyecto SPDY, el IETF (Internet Engineering Task Force) creó un equipo para el desarrollo de una nueva versión del protocolo HTTP, al que más tarde conoceríamos como HTTP/2. Esta nueva versión del protocolo HTTP pretendía, al igual que SPDY, mejorar las comunicaciones en Internet, logrando una reducción de latencia y mejora en la velocidad. El borrador inicial de HTTP/2 se presentó en 2012, pero no es hasta el año 2015 con la publicación del estándar de los RFCs 7540 (HTTP/2) [3] y 7541 (HPACK) [4] cuando los navegadores comienzan a darle soporte. Ese mismo año, SPDY se abandona en favor de HTTP 2.0 debido a que la mayoría de las ventajas que aporta SPDY han sido adoptadas y mejoradas en HTTP 2.0 .

Como respuesta a los aspectos a mejorar de las versiones anteriores para las exigencias de la nueva era en Internet, la IETF comenzó con el desarrollo de la versión HTTP/2 para la estandarización del protocolo y esta se lanzó en 2015 como reemplazo a su predecesora, la versión HTTP/1.1. La nueva versión del protocolo no es una reescritura del mismo, puesto que los métodos, los códigos de estado y la semántica que se utiliza en la nueva versión, es igual que en la versión 1.1. Entre las características que nos ofrece esta versión, nos encontramos con la transmisión binaria de los datos en lugar de la tradicional transmisión en texto plano, la compresión de la cabecera de los mensajes mediante el algoritmo “HPACK”, la multiplexación de los datos en una sola conexión, la priorización de las solicitudes, el uso de la tecnología Server Push que permite al servidor anticiparse a las peticiones del usuario y por último, mejoras en la seguridad con TLS.

A continuación, estudiaremos las principales características [5] que incluye esta versión del protocolo:

Transmisión binaria

Uno de los cambios que introdujo el protocolo SPDY fue el uso del formato binario para la transmisión de datos en lugar de texto sin formato como en las versiones anteriores del protocolo. Dadas las ventajas que ofrece este tipo de transmisión, HTTP/2 también adoptó el mecanismo de transmisión binaria.

Para el usuario es fácil interpretar el texto, pero el formato binario ofrece varias ventajas como la eficiencia a la hora de interpretar el contenido de los mensajes por parte de los sistemas informáticos, una mayor capacidad de compactación a la hora de ser transportado y es menos propenso a errores comparado con los mensajes basados en texto, pues estos a menudo tienen varios problemas con los que lidiar como pueden ser los espacios en blanco, las letras mayúsculas y los finales de línea, entre otros.

Compresión de la cabecera (HPACK)

La especificación de HPACK resuelve uno de los mayores problemas presentes en HTTP/1.1, la compresión de las cabeceras, lo que resulta en un mayor aprovechamiento del ancho de banda al enviar paquetes comprimidos. El algoritmo de compresión de HTTP/2 permite establecer un diccionario entre cliente y servidor. HPACK también hace uso de la codificación Huffman para reducir el tamaño de las cadenas.

El diccionario contendrá una parte estática definida en la especificación, como podemos ver en la Figura 3. Esta tabla de índices proporciona una lista de campos de encabezados de HTTP comunes que todas las conexiones tienen más posibilidades de usar y otra parte dinámica, inicialmente vacía, que se actualiza en base a los valores intercambiados dentro de una conexión específica. Esto permite que el cliente utilice los índices del diccionario para recrear los valores utilizados previamente y que de este modo solo envíen los nuevos campos, reduciendo notablemente el tamaño de las cabeceras. Concretamente, el tamaño de cada solicitud se reduce al usar codificación Huffman estática para los valores nunca antes vistos y una sustitución de índices por valores que ya están presentes en las tablas estáticas o dinámicas a cada lado.

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
15	accept-charset	
16	accept-encoding	gzip, deflate
17	accept-language	

Figura 3. Tabla con los principales índices de la especificación HPACK. RFC 7541

Por su parte, SPDY hacía uso de un algoritmo basado en DEFLATE, específicamente dedicado a las cabeceras, pero este resultó ser vulnerable a los ataques CRIME [6], que podían extraer información de las cookies de las cabeceras comprimidas y secuestrar las sesiones de los clientes.

Flujos Multiplexados

Con la versión 2.0 de HTTP ya no es necesario utilizar varias conexiones TCP paralelas, pues una sola conexión puede transportar cualquier cantidad de flujos de datos, tanto solicitudes como respuestas intercaladas sin que se afecten entre sí. Este nuevo método de transmisión permite que tanto cliente como servidor desglosen las tramas intercambiadas en partes más pequeñas que posteriormente serán reconstruidas en el otro extremo. Este mecanismo ofrece una latencia reducida, mejor rendimiento y mayor aprovechamiento del canal que comparten cliente y servidor.

Aunque este mecanismo de comunicación sea mucho más efectivo y permita el intercambio de mensajes de manera más fluida, todavía no soluciona el problema del bloqueo de línea de cabecera a nivel de TCP. Si un paquete se pierde en la conexión, bloquea todo el flujo a la espera que este sea retransmitido y recibido correctamente en el otro extremo. Al tratarse de un problema ocasionado a nivel de transporte, no es posible evitarlo a nivel de aplicación, pero QUIC propondrá una solución al respecto.

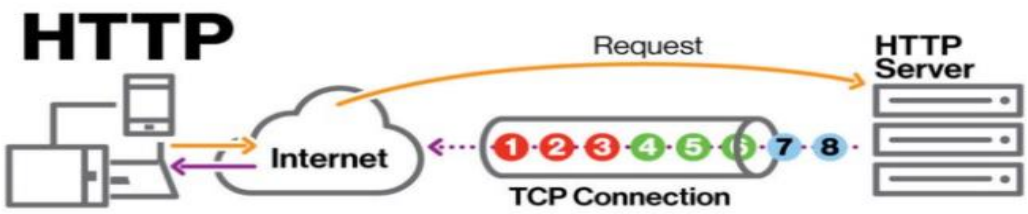

Figura 4. Representación de un flujo multiplexado en HTTP/2. Devopedia

Priorización de flujos

La especificación de HTTP/2 permite que el cliente asigne prioridad a los flujos de datos, incluyendo dicha información en las cabeceras de las tramas que abren dichos flujos, aunque el cliente podrá modificar la prioridad asignada previamente a cualquier flujo cuando crea conveniente.

Este método permite que cliente y servidor intercambien los recursos de manera óptima de acuerdo a las especificaciones de prioridad, ya que permite al cliente recibir las solicitudes de contenido principal y evita que el canal se bloquee con recursos pesados y no críticos que no permiten la descarga de recursos de alta prioridad.

De todos modos, el servidor no está obligado a seguir las indicaciones de prioridad del cliente. Aunque este método puede parecer sumamente útil para la priorización de recursos, en la práctica no se está implementando y de hecho, la versión de QUIC no incluirá este mecanismo.

Server Push

Mejora del método tradicional solicitud-respuesta, en el cual el servidor actuaba de forma pasiva y solo respondía a las peticiones que realizaba el cliente. En este caso el servidor adelanta las peticiones que va a realizar a continuación el cliente con el fin de reducir el número de peticiones por parte de este, es decir, el servidor puede generar una respuesta múltiple para una sola petición del cliente con el fin de que el cliente guarde en la caché la información recibida para no tener que realizar la petición asociada en un futuro.

Cuando consultamos una página web, el servidor envía un HTML y nuestro navegador lo analiza con un algoritmo llamado escáner de precarga, con el que busca los recursos que tiene que pedir al servidor antes de montar la página. Con la nueva técnica de Server Push, podemos adelantar la carga de los recursos para que se descarguen al mismo tiempo que el HTML, ahorrando tiempos de ida y vuelta de solicitud-respuesta entre ambas partes, reduciendo la latencia de la red.

El cliente puede rechazar en cualquier momento los mensajes que esté recibiendo mediante server push por parte del servidor. Además, el server push debe tener en cuenta algunos factores, ya que si el cliente ya tiene el recurso que se está enviando en caché, el método de envío adelantado puede resultar contraproducente.

La carga de los recursos críticos de una página web es el ejemplo más común para el uso de esta técnica de precarga, sin embargo, también puede resultar efectivo en otros ámbitos, como en la transmisión en vivo de baja latencia, como podemos ver en [7], donde se desarrolla una estrategia de k-Push en la que el servidor envía los siguientes k segmentos de vídeo sin que el cliente haga una petición para cada uno de ellos, permitiendo una correcta transmisión sin aumentar la sobrecarga de solicitudes por parte del cliente.

Por otra parte, SPDY incluía dos tipos de flujos iniciados por el servidor [8].

- Server Push: el servidor adelanta las peticiones que va a realizar a continuación el cliente porque prevé que este las va a necesitar.

- Server Hint (indicación): el servidor realiza una sugerencia al cliente para referirse a un recurso que no ha descargado, pero cree que debe descargarse. El servidor no enviará el recurso hasta que el cliente lo solicite de manera explícita.

El método de server Hint no ahorra un viaje de ida y vuelta como en el caso del server push, pero permite al servidor informar de cuales son los recursos críticos que va a necesitar el cliente sin forzar el envío además de permitir al cliente que compruebe si tiene esos recursos en caché, evitando envíos innecesarios. El server hint no se contempla dentro de las especificaciones de HTTP/2 pero es similar a la funcionalidad que incluye con Push request.

Seguridad

Así como no era obligatorio el uso de certificados SSL/TLS para la versión de HTTP/1.1, la especificación original de HTTP/2 tampoco hacía obligatorio el uso de TLS. Sin embargo, los principales navegadores [9] hicieron obligatorio el uso de certificados de seguridad TLS sobre HTTP/2 para cifrar los mensajes. Después rectificaron esta decisión e incluyeron en la especificación el uso obligatorio de TLS 1.2 o superior sobre HTTP/2.

El uso obligatorio de certificados suponía un avance en cuanto a la seguridad a la hora de intercambiar mensajes de forma cifrada, pero suponía un aumento adicional de la latencia para HTTP/2, dado que ahora había que realizar dos handshakes (TCP y TLS) de forma obligatoria para establecer la conexión.

SPDY hacía obligatorio el uso de TLS para cifrar las comunicaciones desde un principio. El protocolo conseguía conexiones cifradas rápidas gracias al uso de la extensión NPN [10]. Pero con el fin de reducir los intercambios de mensajes en el proceso de conexión, se desarrolló la extensión ALPN [11] del protocolo TLS, que permite a la capa de aplicación seleccionar el protocolo que se va a utilizar sobre una conexión segura. Esto mejoraba el intercambio de mensajes en un tiempo de ida y vuelta respecto a la versión anterior, NPN.

Desafíos

Aunque HTTP/2 resuelve los principales problemas de rendimiento de las versiones anteriores, todavía cuenta con algunos problemas relacionados con el protocolo TCP sobre el que se establece, como los retrasos en el establecimiento de conexión TCP + TLS para una transmisión segura, es decir, serán necesarios dos procesos de “handshake” para establecer cada comunicación entre el cliente y el servidor.

Por otro lado, nos encontramos con el problema del bloqueo de cabecera de TCP. Si se produce una pérdida de paquetes en HTTP/2, el protocolo TCP hará uso del mecanismo de retransmisión de pérdida de paquetes y los paquetes perdidos deberán esperar la confirmación de la retransmisión. Al solo disponer de un enlace, se bloquearán todas las solicitudes asociadas a dicha conexión a la espera de la confirmación de la retransmisión.

2.3 QUIC

Dadas las limitaciones con las que nos encontramos debido al protocolo de transporte que se ha utilizado en HTTP desde sus orígenes, ha sido necesario buscar una solución para conseguir una menor latencia y mayor aprovechamiento del ancho de banda. Como TCP ha sido el protocolo de transporte por excelencia en Internet, realizar cambios en el kernel de los SO con el fin de abordar estos problemas resulta poco realista, por tanto, QUIC surge como respuesta a esta necesidad.

QUIC [12] fue propuesto por Google en 2013 y fue definido como un protocolo capaz de llevar varios flujos independientes de manera segura sobre UDP, en lugar de TCP. Además incluía diversas técnicas que estaban dedicadas a reducir el número de viajes de ida y vuelta para mejorar las comunicaciones en Internet. No fue hasta 2016 cuando llevaron la propuesta a la IETF para su proceso de estandarización. De todos modos, el equipo de Google continuaba desarrollando su propuesta de forma independiente [13].

Podemos encontrar varios cambios significativos dentro de este nuevo protocolo, entre ellos, la base sobre UDP como protocolo principal de transporte, la fiabilidad en la entrega de paquetes siguiendo la filosofía utilizada en TCP, las técnicas de corrección de errores, la capacidad de multiplexar varios flujos independientes entre sí para una misma conexión, los identificadores de conexión para la migración de conexión y la seguridad TLS integrada en el protocolo, además de varios cambios y mejoras sobre la propuesta anterior.

QUIC es un protocolo multicapa, principalmente de la capa de transporte, que se basa en UDP y es el protocolo de soporte subyacente en HTTP/3. UDP es un protocolo más ligero que TCP, tanto para referirnos al tamaño de las cabeceras como al intercambio de mensajes que realizan cliente y servidor para comprobar que todo está funcionando correctamente. Por tanto, sustentandose en UDP, el objetivo principal de QUIC es la reducción de la latencia, aunque como veremos, toma la esencia de TCP para lograr un protocolo confiable en los aspectos referentes a pérdidas de paquetes.

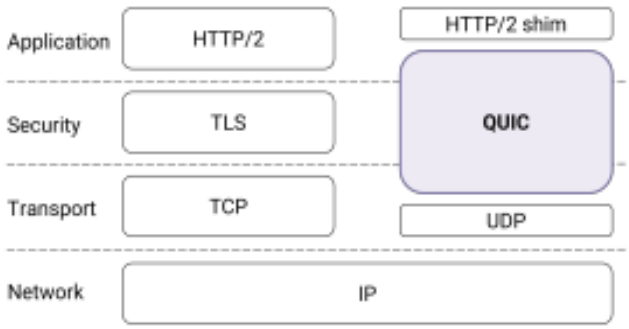

Figura 5. Pila de protocolos con QUIC según el modelo OSI. Chromium

Al basarse en UDP, un protocolo sin conexión, se buscaba que el cliente pudiera realizar peticiones sin realizar un handshake previo con el servidor, lo que significaría mayor velocidad en la comunicación entre ambas partes. Esta es la filosofía que se pretende con QUIC, una vez el cliente haya cacheado la información sobre el servidor y viceversa, el cliente podrá establecer una conexión encriptada 0-RTT sin intercambio de mensajes, es decir solo será necesario realizar un handshake de configuración inicial entre cliente y servidor cuando estos no se hayan comunicado nunca antes.

La principal diferencia entre gQUIC (Google) y QUIC, es que Google desarrolló gQUIC pensando solo en su uso para HTTP, sin embargo, dadas las limitaciones que se presentan en TCP con las tecnologías actuales, la IETF quiere llevarlo más allá y dejar abierta la posibilidad de que se aplique a otros protocolos de la capa de aplicación. Aunque la primera implementación se haya dedicado a HTTP/3, la IETF espera que también sirva de soporte para futuras aplicaciones.

QUIC ha mejorado el diseño original de gQUIC gracias a la colaboración de organizaciones e individuos, con el objetivo compartido de conseguir un Internet mejor, más rápido y más seguro. A finales de 2020, debido a que la versión de QUIC de la IETF había logrado mejoras [14] sobre la versión original , Google empezó a implementar la versión de la IETF en sus navegadores, concretamente la versión preliminar h3-29.

Los navegadores web y los servicios en Internet han estado probando la tecnología durante años, pero el respaldo del IETF es una señal de que el estándar es lo suficientemente maduro como para adoptarse a nivel general. Son cuatro los estándares oficiales a fecha de redacción de este documento.

RFC 8999 - Version-Independent Properties of QUIC

RFC 9000 - QUIC: A UDP-Based Multiplexed and Secure Transport

RFC 9001 - Using TLS to Secure QUIC

RFC 9002 - QUIC Loss Detection and Congestion Control

Según el equipo de desarrollo de QUIC , ya han sido aprobados y están a la espera de la publicación del RFC correspondiente, los documentos de HTTP/3, QPACK, Balanceadores de carga, Operaciones (Aplicabilidad y Manejabilidad) y Extensiones (Negociación de Datagramas y Versiones). Estos están dedicados a definir los últimos detalles del protocolo, así como la publicación oficial del estándar de la tercera versión del protocolo HTTP.

A continuación, vamos a detallar todas las características [15] que incluye esta versión.

Fiabilidad

A pesar de que UDP no ofrece fiabilidad en la entrega de los paquetes, QUIC agrega una capa encima de UDP para garantizar la transmisión, proporcionando retransmisión y seguimiento de paquetes y control de congestión del mismo modo que TCP. El algoritmo que se emplea en QUIC para el control de la congestión es similar a la versión CUBIC que utiliza TCP [16].

Con el objetivo de recuperar paquetes perdidos sin esperar a una retransmisión, QUIC actualmente trabaja con un esquema FEC (Forward Error Correction) sencillo basado en XOR. En el flujo de paquetes, se envía un paquete FEC que contiene la paridad de los paquetes de un grupo determinado. Si uno de los paquetes del grupo se pierde, el contenido de este se puede recuperar del análisis del paquete FEC y del resto de paquetes del grupo, es decir, no es necesaria la retransmisión de la información original. El emisor decide cuándo enviar paquetes FEC para optimizar la transmisión en distintos escenarios.

Multiplexación - Flujos

Se mejora la multiplexación que ofrecía la versión 2 del protocolo HTTP, ya que con QUIC es posible introducir varios flujos independientes en una misma conexión para el intercambio de información. Con este modelo se resuelve parcialmente el problema de bloqueo del canal ante la pérdida de paquetes que sucedía con TCP, es decir, la pérdida de un paquete en un canal determinado ya no afecta al resto de canales, que pueden seguir la transmisión con normalidad.

Debido a este nuevo modelo, los paquetes pueden ser recibidos fuera de orden para los distintos flujos, pero un mismo flujo garantiza que los paquetes son recibidos en orden. Los extremos utilizarán el identificador correspondiente a cada flujo y el desplazamiento asociados a cada paquete para colocar los paquetes en orden.

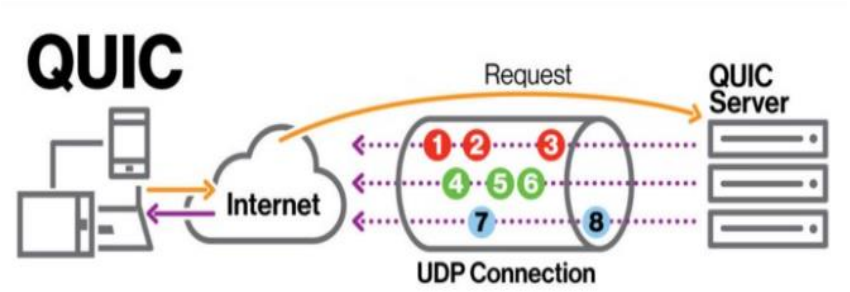

Figura 6. Representación de un flujo multiplexado en HTTP/3 con QUIC. Devopedia

Identificador de conexión

La conexión se negocia entre cliente y servidor mediante QUIC por un puerto UDP y una dirección IP, pero una vez se ha establecido la conexión, esta se asocia con un "identificador de conexión". La función principal de este identificador es asegurarse que cualquier cambio en la dirección de protocolos inferiores no provoque que los paquetes sean enviados a un extremo diferente. Con esta ventaja, las conexiones pueden moverse entre diferentes IPs de forma que no podría TCP.

El uso de identificadores de conexión (ID) permite que las conexiones cambien sus direcciones (direcciones IP y puertos) en casos de migración a otra red. QUIC se basa en que las conexiones entre cliente y servidor conservan un identificador ID, por tanto, pueden sobrevivir a cambios de IP y a cambios de traducciones NAT, ya que la ID de la conexión permanece constante durante estas migraciones. Esto con TCP no era posible, ya que las conexiones estaban identificadas con 2 pares IP-puerto (fuente y destino) y era necesario volver a realizar el handshake de configuración (TCP y TLS). De este modo, con QUIC es posible realizar un cambio de red manteniendo activa la conexión cifrada**,** ya que el cliente mantiene la misma clave de sesión para dicha conexión.

Compresión de cabecera (QPACK)

En cuanto a la compresión de las cabeceras con HTTP/3, QPACK [17] sigue la misma filosofía que ya introducía la versión 2 del protocolo con su algoritmo de compresión HPACK, mantiene las tablas, estática y dinámica, y la codificación Huffman.

La modificación que sufre la nueva versión del algoritmo es que ha sido rediseñado para permitir a los extremos recibir paquetes desordenados. Debido al bloqueo de línea de cabecera presente en TCP, no podía darse el caso de que el algoritmo HPACK tuviera que tratar con paquetes desordenados, ya que había que esperar a que el paquete fuera retransmitido y recibido correctamente en el otro extremo.

Priorización de flujos

HTTP/3 ya no cuenta con el modelo de priorización que se había añadido en HTTP/2. Dado que la nueva versión trabaja con varios flujos multiplexados para una misma conexión, este mecanismo no resultaba tan efectivo como podría serlo en la versión 2, donde solo existía una conexión y si que era necesario establecer un orden de prioridad.

Server Push

Dado el poco interés que los servidores tenían sobre implementar la técnica de server push, el equipo de desarrollo de gQUIC estaba planteando eliminarla para su versión del protocolo. Según un estudio realizado a finales del 2020, el 99.95% de las conexiones nunca crearon un flujo para implementar esta funcionalidad. Además, de las pocas conexiones que sí hacían uso de Server push, solo el 40% resultaban efectivas, el resto eran incorrectas o ya estaban en la caché del cliente.

A pesar del estudio anterior, la IETF sí que ha decidido implementar el método de Server Push en HTTP/3 y de hecho, es similar al descrito en la versión 2.0, con algunas pequeñas modificaciones. En esta versión, el cliente puede establecer un número máximo para limitar las veces que el servidor realiza la operación. Aunque este método mejora la comunicación evitando el mensaje de solicitud del cliente, todavía cuenta con los mismos problemas que la versión anterior, dado que el servidor no es capaz de determinar con total grado de exactitud si un recurso debería ser enviado por este método o no.

Seguridad

En lo referente a la seguridad, QUIC hace uso por defecto de la versión TLS 1.3 y por tanto, es obligatorio con el objetivo de dificultar que dispositivos intermedios manipulen o detecten el tráfico. De este modo, no cabe la posibilidad de que haya una conexión sin cifrado. Como QUIC lleva integrado TLS, permite cifrar los mensajes sin intercambiar claves privadas entre cliente y servidor después de haber realizado el handshake de configuración inicial.

Desafíos

Dado que TCP ha sido el protocolo de transporte principal en numerosas aplicaciones durante mucho tiempo, se temía que los sistemas operativos y el software no estuvieran optimizados para el uso de UDP. Sin embargo, se ha podido comprobar que la eficiencia computacional [18] para QUIC puede ser la misma que para TCP. Con la estandarización e implementación final de QUIC se espera que veamos mejoras enfocadas en estos aspectos por parte de los desarrolladores.

2.4 Despliegue actual

Como hemos mencionado anteriormente, en 2016 la IETF publicó el primer borrador para el desarrollo y la estandarización del protocolo QUIC. Esto fue solo un año después de presentar el estándar HTTP/2 basado en el protocolo SPDY de Google. El objetivo de ambas propuestas ha sido el mismo, mejorar la velocidad, los tiempos de carga y las latencias en la web además de añadir notables mejoras a la seguridad.

En las siguientes imágenes, podemos observar datos interesantes en relación al despliegue actual de ambas tecnologías.

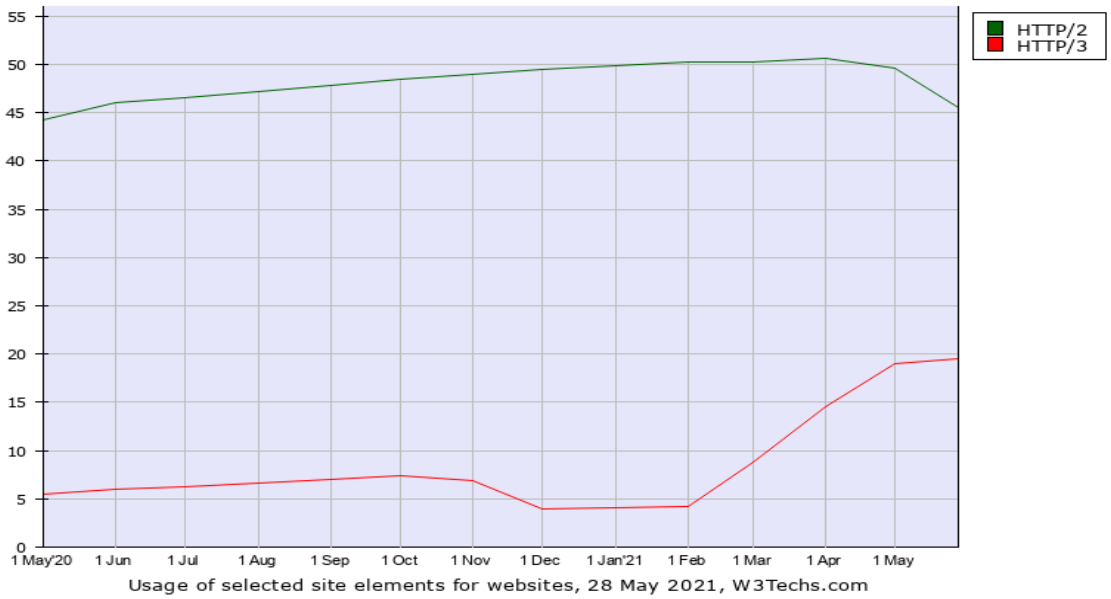

Figura 7. Evolución del uso de HTTP/2 y HTTP/3 en Internet. W3Techs.com

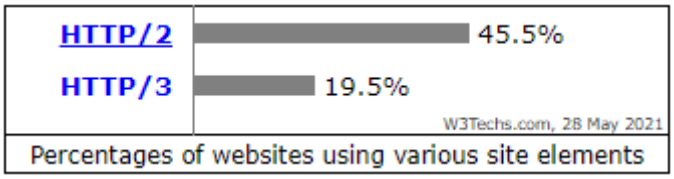

Figura 8. Despliegue actual de HTTP/2 y HTTP/3. W3Techs.com

Hoy en día, la versión HTTP/2 ya es usada por el 45.5% de las páginas web. Al estar más extendida, encontramos muchas más empresas conocidas como Amazon, Yahoo, Wikipedia, Zoom o Paypal.

Por su parte, cerca del 20% del tráfico en la web hace ya uso de HTTP/3, entre los sitios más conocidos que hacen uso de este protocolo, nos encontramos con grandes empresas como Google, Facebook o Shopify. Al tratarse de sitios web con un gran tráfico en Internet, suponen una parte importante del tráfico total.

3. Estudio de latencias para las versiones de HTTP

A continuación vamos a estudiar el intercambio de mensajes que se realizan en HTTP para todas sus versiones, revisando el inicio de la conexión, el flujo de datos entre ambas partes y el cierre de la conexión.

3.1 HTTP/1.1

En esta versión del protocolo, como hemos visto anteriormente, los datos no van cifrados y por tanto es el menos seguro de todos, puesto que se podrían interceptar los datos e interpretarlos con facilidad.

El handshake inicial del protocolo TCP consiste en un proceso de tres pasos:

1. El cliente informa al servidor que quiere iniciar la conexión con un segmento de sincronización SYN en el que se encuentra el número de secuencia elegido por el cliente.

2. A continuación el servidor responderá con un segmento con los bits de SYN y ACK activos. El ACK sirve para confirmar el segmento que ha recibido por parte del cliente y SYN indica el número de secuencia por el que deben comenzar los mensajes.

3. Por último, el cliente envía un ACK de reconocimiento como respuesta al mensaje del servidor para establecer la conexión y comenzar con el intercambio de datos.

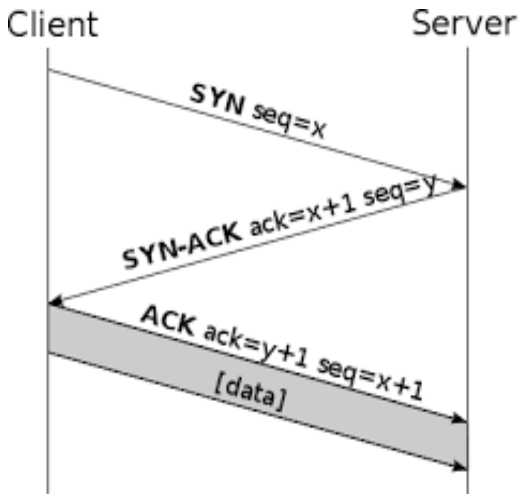

Figura 9. Establecimiento de conexión TCP. Wikipedia

Una vez iniciada la conexión, cliente y servidor podrán comenzar a intercambiar datos. El intercambio de mensajes está basado en el modelo solicitud respuesta, donde

el servidor servirá las peticiones a medida que las vaya haciendo el cliente y dependiendo de si se emplea una conexión diferente para servir cada solicitud o si se emplea la misma conexión para servir todas las solicitudes, estaremos haciendo uso de conexiones no persistentes y conexiones persistentes, respectivamente.

A parte de las conexiones persistentes y no persistentes, podemos utilizar técnicas que mejoren el aprovechamiento de la conexión, como pueden ser las conexiones paralelas o el pipeline. Cada una tiene sus ventajas respecto a la otra, por ejemplo, las conexiones paralelas permiten obtener antes los recursos de una página web, ya que se emplean varias conexiones TCP para solicitar los recursos. Por otro lado, con el pipelining se consigue un mayor aprovechamiento del canal, realizando varias peticiones consecutivas sobre la misma conexión TCP.

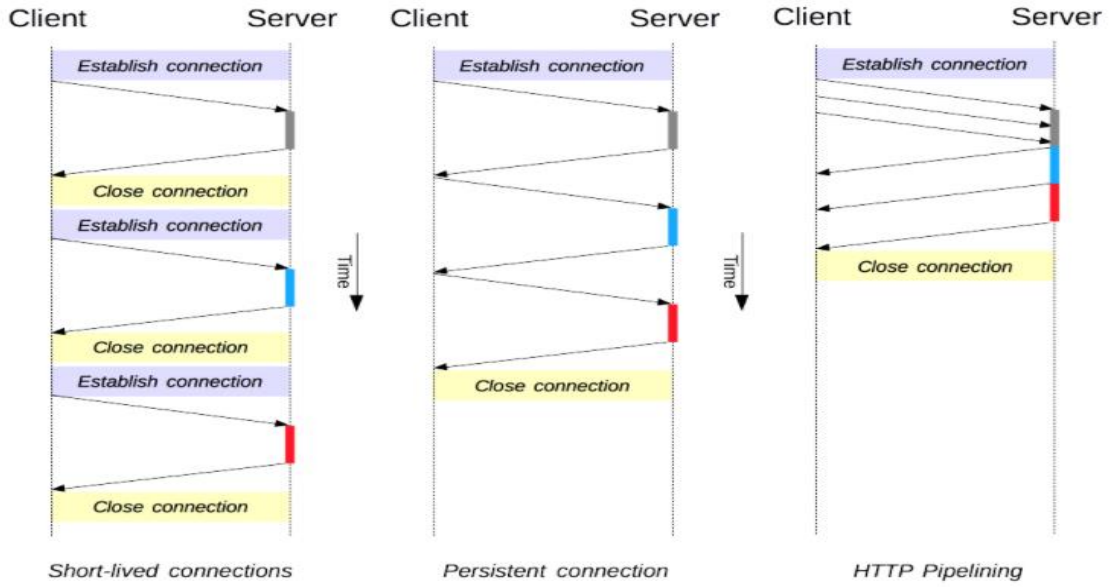

Figura 10. Distintos métodos para conexiones TCP. MDN Mozilla

Una vez el cliente haya recibido todos los datos que ha solicitado al servidor, enviará un ACK de confirmación para informar al servidor. En este punto, cualquiera de las dos partes puede iniciar el cierre de la conexión con un mensaje FIN, el cual no tiene datos, solo cabecera. La otra parte contestará a este mensaje con un ACK de reconocimiento y con otro mensaje FIN. Por último, el agente que inició el cierre de la conexión confirmará el mensaje con otro ACK.

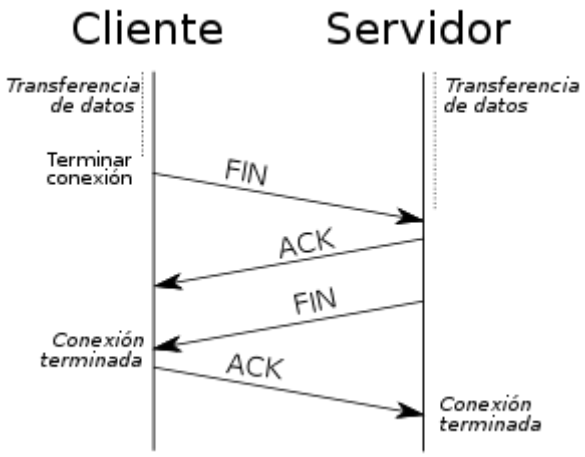

Figura 11. Cierre de conexión TCP. Wikipedia

3.2 HTTP/2

Esta versión del protocolo también está basada en TCP, por tanto, mantiene el handshake de tres pasos para el establecimiento de la conexión y el de cuatro pasos para el cierre. Sin embargo, cuenta con un protocolo adicional, encargado de cifrar los datos, por tanto, después de haber establecido la conexión TCP, se establece la conexión TLS.

TLS

Los usuarios de Internet son conscientes de la cantidad de amenazas que se pueden encontrar al navegar por la web. Para adaptarse a las necesidades de Internet, fue necesario reforzar la seguridad cifrando los datos, para evitar que terceras personas pudieran acceder a la información que intercambian cliente y servidor, como contraseñas, información bancaria o datos personales de los usuarios.

El protocolo criptográfico que encripta los datos y autentifica una conexión cuando se mueven los datos en Internet es TLS [19][20], o Transport Layer Security. TLS es en realidad sólo una versión más reciente del original SSL y corrige algunas vulnerabilidades de seguridad en los protocolos SSL anteriores. TLS hace uso de la criptografía asimétrica para compartir de forma segura la clave que utilizarán ambas partes durante la comunicación con criptografía simétrica. Con esta técnica se consigue aprovechar las ventajas de ambos protocolos, esto es, la mayor seguridad que ofrece la criptografía asimétrica con la velocidad que proporciona la criptografía simétrica.

Pasos para establecer la conexión cifrada.

1. El cliente inicia la comunicación enviando un mensaje “ClientHello” al servidor. El mensaje incluirá la versión de TLS que soporta el cliente, el método de compresión y los conjuntos de cifrado compatibles, así como una cadena de bytes aleatorios, conocidos como “client random”.

2. Como respuesta al mensaje del cliente, el servidor enviará un mensaje “ServerHello”, que contendrá el certificado SSL del servidor, el cifrado escogido por parte del servidor de entre el conjunto y el “server random”, otra cadena aleatoria de bytes que son generados por el servidor. Con el certificado digital, el cliente podrá verificar la identidad del servidor y además, este contendrá la clave pública del mismo.

3. A continuación, el cliente envía el mensaje “ClientKeyExchange” que contendrá la PreMasterSecret (generada a partir del método de cifrado escogido) cifrada a partir de la clave pública que ha obtenido del servidor. A continuación, envía un mensaje “ChangeCipherSpec” indicando que ambas partes ya pueden hacer uso de criptografía simétrica. Por último, el cliente envía un mensaje “ClientFinished” indicando que ya ha acabado su parte de la autenticación.

4. El servidor desencripta el mensaje Key Exchange que ha recibido por parte del cliente utilizando su clave privada de criptografía asimétrica. Y en este punto ambas partes generan la Master SecretKey a partir de los valores aleatorios (Client y Server Random) generados anteriormente, la única clave que se usará a partir de ahora para el intercambio de datos utilizando criptografía simétrica. El servidor envía un mensaje (“Change Cipher Spec”) para indicar que a partir de ahora va a usar criptografía simétrica. Y un mensaje (“Server Finished”) indicando que ya ha terminado su parte de la negociación.

El cifrado asimétrico es más seguro, pero requiere más tiempo de procesamiento. De este modo, para que cliente y servidor puedan intercambiar datos encriptados de forma rápida y segura, se establece una conexión mediante criptografía asimétrica para que una vez exista un medio seguro por el que intercambiar información, puedan acordar una misma clave y continuar con criptografía simétrica.

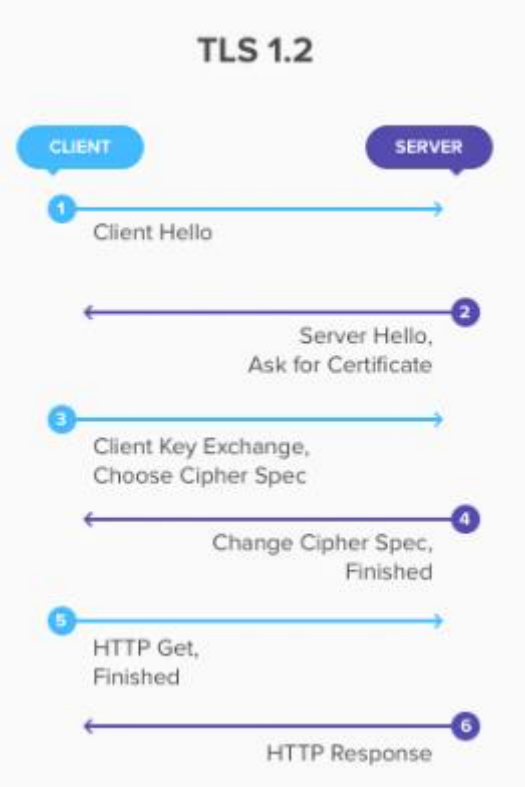

Figura 12. Establecimiento de conexión TLS 1.2. CDN77

Hasta este punto, la conexión con HTTP/2 no difiere en nada respecto a lo que sería una conexión cifrada de HTTP/1.1 con HTTPS. Sin embargo, la especificación de HTTP/2 obliga a incluir un mensaje “GOAWAY” previo al cierre de la conexión TLS, una vez que ambas partes hayan terminado el intercambio de mensajes, para indicar que ya no se aceptan más datos y poder gestionar adecuadamente el cierre de los flujos. Cuando el cliente responda con el ACK de confirmación correspondiente al mensaje de cierre GOAWAY, el servidor continuará con la terminación de la conexión TLS.

Para cerrar la conexión TLS, el servidor enviará un mensaje Encrypted Alert para informar al cliente que va a cerrar dicha conexión, porque pretende cerrar la conexión TCP. Este mensaje se envía para informar a la otra parte de que se va a dejar de cifrar los datos, de este modo se evitan ataques de secuestro de sesión. Cuando reciba un ACK del cliente con la confirmación al mensaje Encrypted Alert, comenzará el proceso de cierre de TCP de cuatro pasos, con la diferencia de que el cliente, antes de enviar su mensaje FIN para advertir del cierre de la conexión, tendrá que enviar un mensaje Encrypted Alert del mismo modo que lo hizo el servidor para terminar la sesión TLS.

3.3 QUIC

QUIC, al basarse en UDP, un protocolo no orientado a la conexión, no necesita realizar un handshake de configuración inicial como lo hacían las versiones anteriores que funcionaban sobre TCP. QUIC solamente necesita enviar un mensaje con un identificador de conexión para establecer una relación cliente-servidor e inmediatamente puede empezar a realizar las solicitudes, sin esperar a la confirmación por parte del servidor.

Sin embargo, aunque QUIC pueda intercambiar mensajes sin establecer una conexión previa, su especificación obliga a que todos los mensajes de este tipo vayan cifrados como mínimo con la versión de TLS 1.3, es decir, su última versión. De este modo, la primera vez que se conecten un cliente y un servidor, deberán configurar el protocolo de seguridad para obtener las claves y poder enviar los datos cifrados.

TLS 1.3

TLS 1.3 es la nueva versión del protocolo y fue aprobada en 2018 por la IETF. Uno de los objetivos de TLS 1.3 es reducir la latencia durante el establecimiento de la conexión. Esta versión introduce dos avances significativos en este aspecto, los modos de conexión con 1-RTT y 0-RTT.

El proceso para establecer la conexión con el modo 1-RTT sigue la misma filosofía que las versiones anteriores, pero reorganiza el contenido de los mensajes para reducir el tiempo de ida y vuelta. Estos son los pasos necesarios para establecer la conexión con TLS 1.3

1. Del mismo modo que con la versión 1.2, el cliente inicia la conexión con un mensaje “Client Hello”. El cliente envía los conjuntos de cifrado compatibles, y presupone cual es el protocolo que el servidor va a elegir a continuación y además de un mensaje “Key Share” con la llave específica para ese protocolo.

2. El servidor responde con un mensaje “Server Hello”, que contiene el algoritmo de cifrado escogido para la comunicación de la lista que ha enviado el cliente, seguido de un “Key Share” que utilizará el cliente para generar la clave simétrica, un mensaje indicando el certificado digital que utiliza el servidor y el mensaje “Server Finished”, para indicar que ya ha terminado su parte del proceso de conexión.

3. Por último, el cliente comprueba el certificado del servidor, generará la misma clave simétrica que el servidor y enviará el mensaje “Client Finished” para finalizar el proceso. A partir de este punto, los datos se enviarán de forma segura, cifrados con criptografía simétrica.

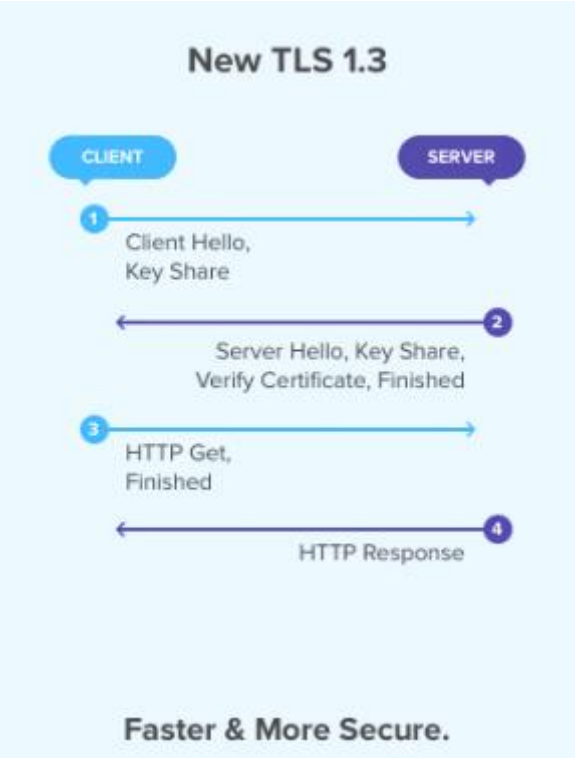

Figura 13. Establecimiento de conexión TLS 1.3. CDN77

Con la versión 1.3 de TLS solo hace falta un RTT para establecer una conexión segura, en lugar de los dos RTT que requerían las versiones anteriores. Aplicando esto al intercambio de mensajes de QUIC, puede no tener gran relevancia, pues cliente y servidor solo establecerán la conexión la primera vez, sin embargo, TLS 1.3 puede aplicarse a las conexiones HTTPS y HTTP/2 para reducir en un RTT el tiempo de conexión TCP + TLS que tiene que hacerse cada vez que cliente y servidor quieran intercambiar datos.

El gran beneficio que aporta esta versión al protocolo QUIC es su modalidad de conexión con 0-RTT, que permite enviar datos cifrados de forma instantánea.

0-RTT

Dado que cerca del 40% de las conexiones a Internet son reanudadas, el mecanismo que incorpora la versión 1.3 de TLS, 0-RTT[23], nos permite el intercambio de mensajes cifrados de manera inmediata. Cuando el cliente y el servidor ya han establecido una conexión cifrada con anterioridad, ambas partes comparten una PSK que pueden mantener para intercambiar mensajes cifrados en futuras conexiones. Esto permite reanudar las conexiones con los parámetros que fueron configurados por ambas partes en el handshake de la primera conexión, evitando que se repita el proceso en conexiones futuras.

Esto supone una reducción importante de la latencia en la comunicación entre ambas partes, sin embargo, el hecho de mantener siempre la misma clave para la comunicación compromete algunos aspectos en la seguridad de la comunicación.

Con la comunicación mediante 0-RTT, los mensajes son vulnerables a ataques de repetición. Si un atacante consigue los datos que el cliente está enviando al servidor, el atacante podrá replicar esa información al servidor, ya que este no tiene forma de averiguar si los datos proceden del cliente original.

Existen varios mecanismos para evitar los ataques de repetición, entre ellos, Single-Use Tickets o Client Hello Recording, aunque estos pueden suponer una gran carga de almacenamiento global. Por otra parte, en la práctica se está implementando un mecanismo más sencillo, aunque un poco más limitado. Este mecanismo consiste en sólo permitir operaciones idempotentes bajo la comunicación 0RTT, impidiendo así que se repliquen operaciones que podrían comprometer a cualquiera de las dos partes.

El uso de 0-RTT sigue en desarrollo y, por tanto, no es de uso obligatorio, por lo que cualquiera de las dos partes puede desactivarlo y establecer la conexión 1-RTT que incluía la versión 1.3.

En la siguiente figura, se representan de manera sencilla los conceptos que acabamos de exponer.

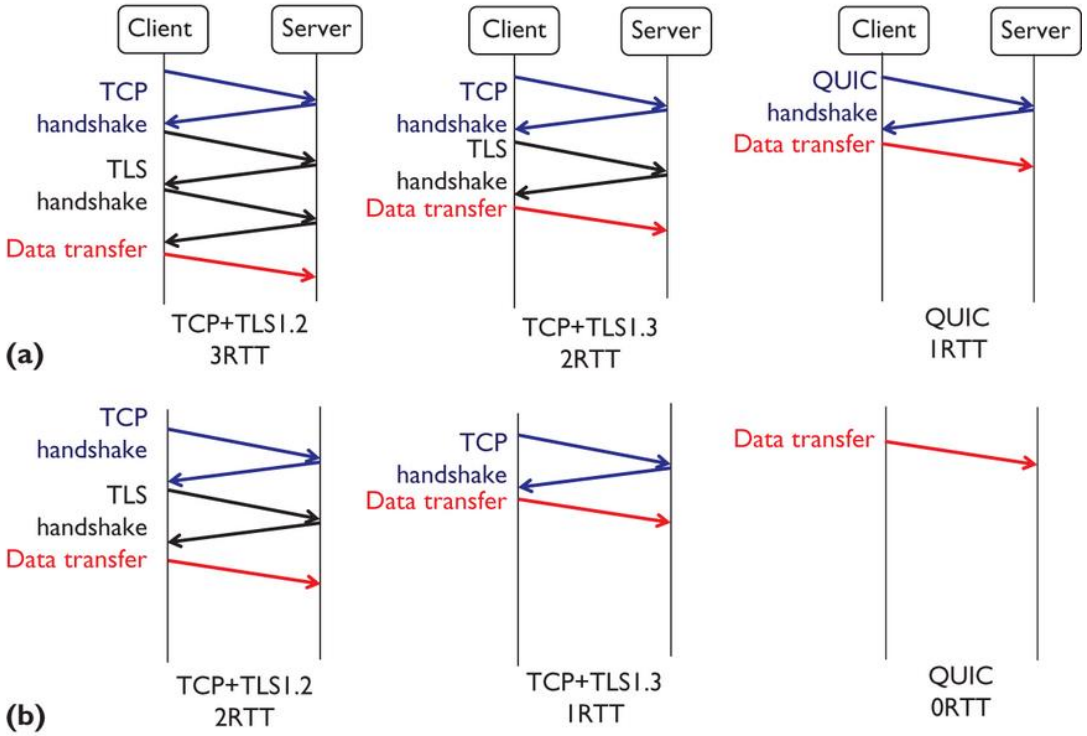

Figura 14. Establecimiento de conexión para los distintos protocolos a) para la primera conexión b) subsecuentes conexiones. Microsoft

4. Pruebas de rendimiento

Ahora que hemos estudiado las principales características que introducen las nuevas versiones del protocolo HTTP, vamos a realizar algunas pruebas de rendimiento sobre distintos escenarios para comprobar cómo se comporta cada versión ante distintas simulaciones en la configuración de los parámetros del enlace.

En primer lugar, crearemos una red privada virtual con la herramienta VirtualBox. El proceso de instalación y configuración de las máquinas virtuales se muestra detalladamente en el Anexo A. El servidor web tiene instalado un sistema Linux con distribución Ubuntu 18.04 LTS y en él instalaremos el servidor web Apache con HTTP/2. El cliente tiene las mismas características que la máquina del servidor y lo utilizaremos para conectarnos al servidor web. El proceso de configuración del servidor Apache con HTTP/2 viene en el Anexo B.

En el primer escenario, el cliente va a realizar una petición HTTP sobre la página web del servidor. Realizaremos distintas peticiones variando los parámetros de configuración en el enlace que conecta a ambas máquinas y compararemos los resultados de la versión HTTP/1.1 con HTTP/2 y con la funcionalidad de Server push de esta última.

Seguidamente, replicando los ejemplos de Akamai y de Golang, disponibles en https://http2.akamai.com/demo y https://http2.golang.org/gophertiles , respectivamente, crearemos nuestro propio ejemplo de carga de imágenes para comparar los resultados obtenidos para peticiones con las versiones de HTTP/1.1, HTTP/2 y de nuevo, con la funcionalidad Server Push.

Por último, utilizaremos la máquina cliente de Linux, con conexión a Internet, para acceder a los sitios web que hacen uso de HTTP/3 y así, poder comparar los tiempos de carga respecto a HTTP/2 para distintas configuraciones en la descarga.

Además, queremos destacar que cuando nos referimos a HTTP/1.1, en realidad estamos realizando peticiones sobre HTTPS para que los mensajes vayan cifrados. De este modo, la comparación con HTTP/2 y HTTP/3 será equitativa, ya que estas dos versiones exigen que los datos vayan cifrados.

Estructura de las pruebas

Para la realización de las pruebas tomaremos como medida de referencia el tiempo de carga completa de la página. En concreto, tomaremos 10 medidas para cada una de las pruebas y las promediaremos para obtener una aproximación del tiempo medio de carga de la página.

Casos de prueba

Después de haber experimentado con un alto número de posibilidades y de combinaciones para cada prueba de ancho de banda, latencia y pérdida de paquetes, en el apartado de resultados mostramos los que nos han parecido más significativos e interesantes para realizar la comparación. Estos se recogen a continuación:

Ancho de banda del enlace:
800 Mbps - 300 Mbps - 100 Mbps - 10 Mbps - 4 Mbps - 1 Mbps - 500 kbps

Latencia del enlace:
25ms - 50ms - 100ms - 500ms -1000ms

Pérdida de paquetes:
1% - 5% - 10% - 25% - 50%

Herramientas para la simulación

1. WonderShaper

Para limitar el ancho de banda del canal, vamos a utilizar un programa llamado Wondershaper, el cual es exclusivo para sistemas Linux.

Su instalación es muy simple y la podemos completar siguiendo los siguientes pasos:
sudo apt-get install wondershaper
git clone https://github.com/magnific0/wondershaper.git
cd wondershaper
sudo make install

Una vez instalado, ya estará listo para usarse.

Como en nuestra red virtual hemos configurado el cliente para que no tenga acceso a Internet, lo descargamos en el lado del servidor, aunque podría estar instalado en cualquiera de las dos partes.
Las opciones para el comando de uso solo son estas tres.

sudo wondershaper -a <InterfazDeRed> -d <V-Descarga> -u <V-Subida>

-a : Nombre de la tarjeta de red.
-d: Velocidad de descarga (kbps).
-u: Velocidad de subida (kbps).

En nuestro caso, al estar instalado el programa en el lado del servidor, solo necesitaremos las opciones de la interfaz de red y de la velocidad de subida.

Cada vez que queramos borrar las opciones para dejar el enlace por defecto o cada vez que queramos modificar de nuevo el ancho de banda, tendremos que lanzar la siguiente orden:

sudo wondershaper -ca <InterfazDeRed>

Como medida adicional para comprobar que el ancho de banda se estaba limitando a la velocidad establecida, utilizamos la herramienta de google para medir la velocidad de subida y de descarga, como podemos apreciar en la siguiente imagen.

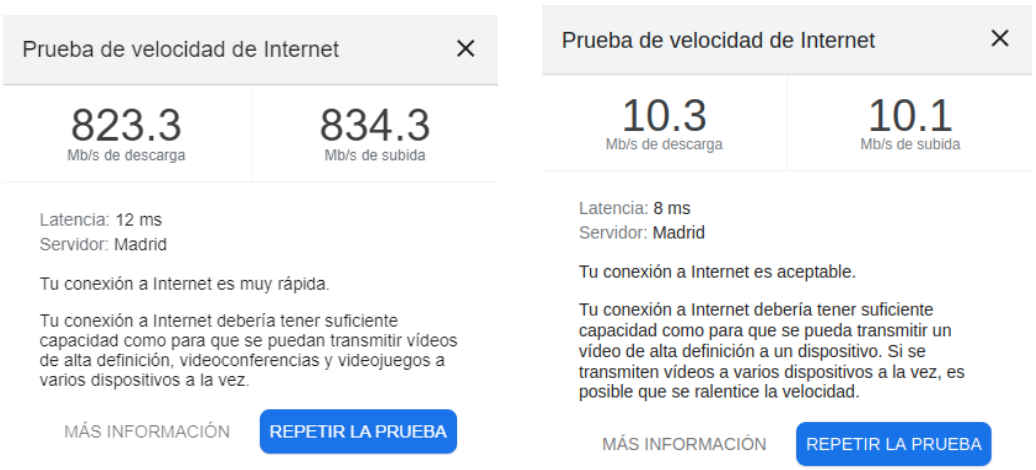

Figura 15. Prueba de velocidad de conexión a) sin limitar b) limitando a 10Mbps.

2. Linux Traffic Control (TC)

Traffic Control es una utilidad de Linux que ofrece la posibilidad de crear simulaciones en la transmisión de paquetes. Las funcionalidades que nos interesan a nosotros son limitar el ancho de banda, añadir latencia y simular la pérdida de paquetes, pero además podríamos simular paquetes repetidos, paquetes dañados o paquetes desordenados.

Durante nuestras pruebas, la herramienta no nos permitía limitar el ancho de banda al mismo tiempo que añadimos latencia y pérdida de paquetes, por ese motivo, decidimos utilizar la herramienta WonderShaper que hemos presentado antes para controlar dicha funcionalidad.

El comando es un poco más complejo que el anterior por la cantidad de funciones que tiene esta herramienta, pero nosotros solo tenemos que modificar tres parámetros:

sudo tc qdisc add dev <InterfazDeRed> root netem delay <latencia (ms)> loss <pérdida (%)>

-dev: Nombre de la tarjeta de red.
-delay: Añade latencia a la transmisión (ms).
-loss: Simula la pérdida de paquetes (%).

Las dos opciones pueden utilizarse de manera conjunta o de manera individual. Tanto los “ms” como el “%” deben indicarse explícitamente en el comando, de lo contrario, el sistema no reconocerá la orden.

Y del mismo modo que con WonderShaper, para volver a la configuración por defecto o para añadir una configuración diferente, primero debemos eliminar la regla actual, por tanto, utilizamos el comando:

sudo tc qdisc del dev <InterfazDeRed> root

Por último, destacar que ambas herramientas no se pueden utilizar al mismo tiempo en un sistema, por lo que utilizaremos Wondershaper en el lado del servidor y la herramienta TC en el lado del cliente.

4.1 Primer escenario

El escenario que planteamos en esta primera parte consiste en que el cliente va a realizar una petición HTTP sobre la página web del servidor, la cual contiene un archivo html, un css y una imagen. Aunque se trate de una estructura muy sencilla, nos ha servido como primer contacto para entrar en contacto con las herramientas de simulación y para determinar los valores de configuración adecuados para las pruebas.

Los dos primeros escenarios hemos querido relacionarlos con la carga de una imagen, aunque el tipo de pruebas para cada uno de ellos sea diferente. Además, una imagen nos permite tener una retroalimentación visual de cómo avanza el estado de la carga del sistema, lo que resulta de gran ayuda a la hora de realizar este tipo de pruebas.

A continuación podemos ver la carga de la imagen de 450x450px utilizada para las pruebas en el primer escenario y que mostramos principalmente para realizar la comparación con la segunda prueba que realizaremos.

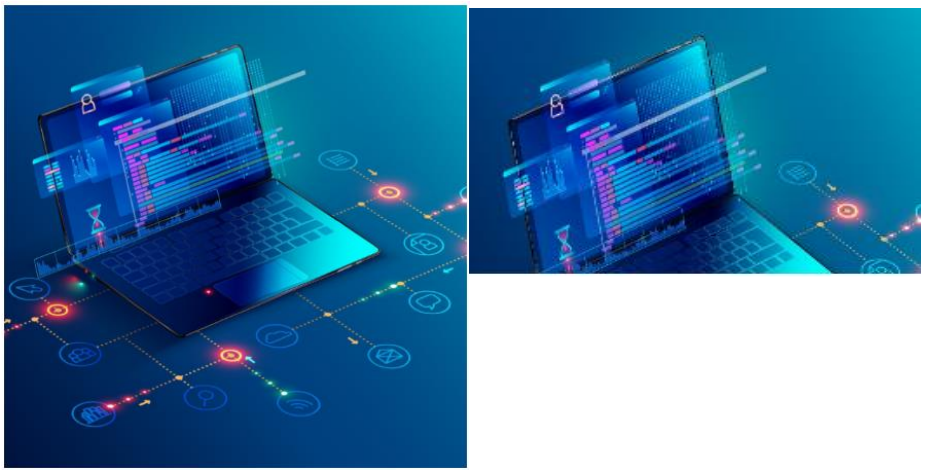

Figura 16. Carga de imagen completa con HTTP.

Limitando ancho de banda, sin latencia y sin pérdidas de paquetes

En la primera prueba de todas, solo vamos a limitar el ancho de banda del enlace para simular distintas velocidades de conexión al servidor y no vamos a añadir latencia ni pérdida de paquetes. Además, como podemos ver en la siguiente imagen en la que hemos realizado un ping a la máquina del servidor, se aprecia que la latencia es prácticamente cero y tampoco tenemos pérdidas de paquetes debido a que ambas máquinas están trabajando en la misma red.

```
Haciendo ping a 192.168.56.106 con 32 bytes de datos:
Respuesta desde 192.168.56.106: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.56.106: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.56.106: bytes=32 tiempo<1m TTL=64
Respuesta desde 192.168.56.106: bytes=32 tiempo<1m TTL=64

Estadísticas de ping para 192.168.56.106:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 0ms, Máximo = 0ms, Media = 0ms
```

Figura 17. Ping entre máquinas de la misma red.

En este primer escenario, para las pruebas con un gran ancho de banda en la comunicación, no hemos encontrado diferencias significativas respecto al tiempo de carga para cada caso de estudio. En los tres casos, la diferencia en los tiempos de carga solo difiere algunos milisegundos.

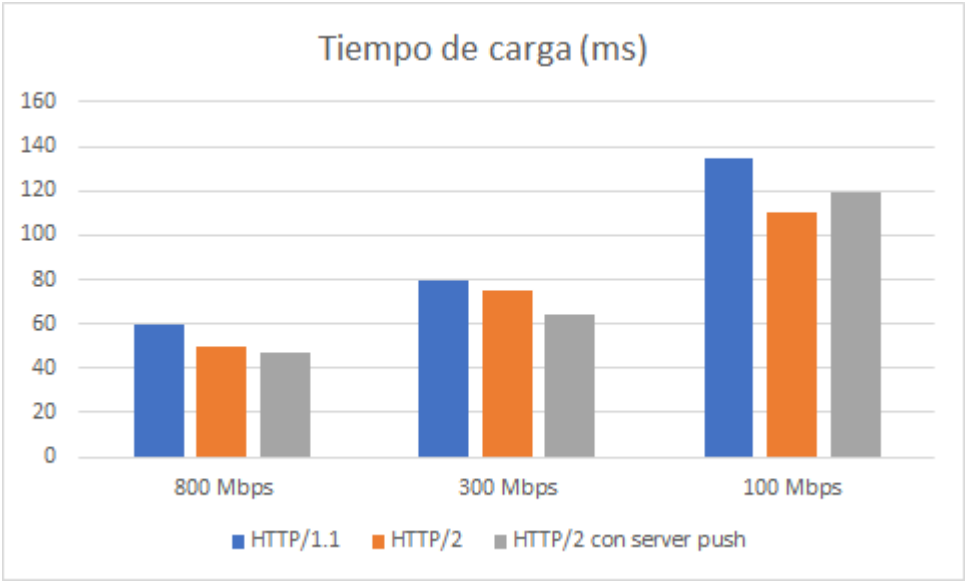

Gráfico 1. Tiempo de carga limitando ancho de banda - Escenario 1

En cuanto a las pruebas para las conexiones de 10 Mbps a 500 Kbps, como podemos apreciar en la siguiente figura, tampoco encontramos grandes diferencias. Al igual que en el caso anterior, solo hay una diferencia de milisegundos, incluso para el caso de la conexión más baja.

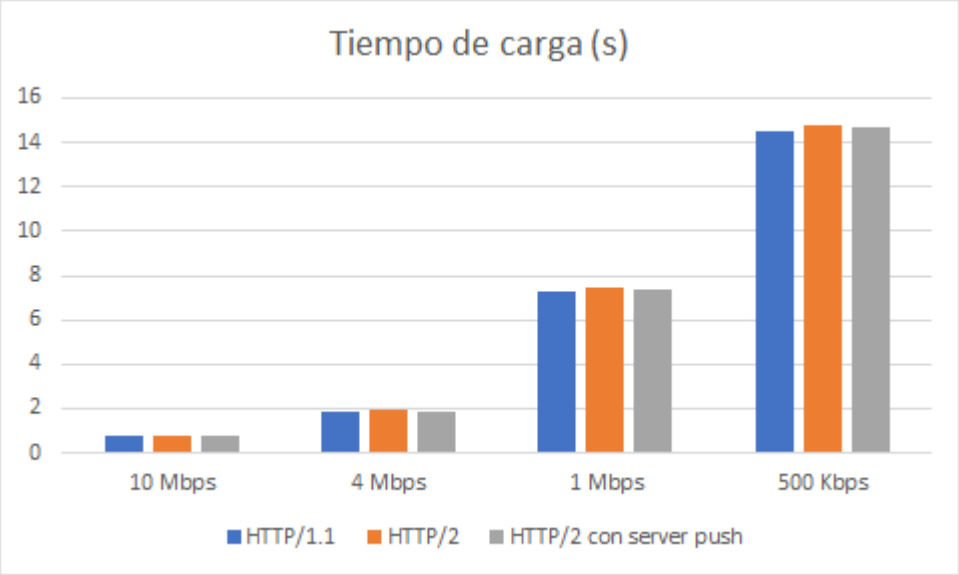

Gráfico 2. Tiempo de carga limitando ancho de banda (2) - Escenario 1

El hecho de que no encontremos diferencias en ninguno de los casos se debe a que solo realizamos tres peticiones para cargar la página web, que además es muy básica, por lo que en este escenario no se puede aprovechar las ventajas que nos ofrece HTTP/2 ni con server push.

Añadiendo latencia con velocidad de 300Mbps y sin pérdida de paquetes

Como 800Mbps es la conexión máxima de la que disponíamos a la hora de realizar las pruebas y comprobamos que esta podía ser algo inestable dependiendo del momento de realizar las pruebas, decidimos establecer el ancho de banda en 300Mbps para poder asegurarnos de que todas las pruebas se realizaban bajo las mismas condiciones.

Como se aprecia en la siguiente gráfica, los tiempos de carga para los casos de baja latencia no tienen un gran impacto, sin embargo, a medida que la latencia va aumentando podemos ver que HTTP/2 se comporta mucho mejor que HTTP/1, llegando incluso a conseguir cargar la página en la mitad de tiempo para latencias de 500ms y superiores. Por otro lado, la funcionalidad de server push, con la cual conseguimos enviar el css y la imagen antes de que el cliente lo solicite, parece no obtener mejores resultados que HTTP/2. En un principio, esperábamos que mejorara el tiempo de carga respecto a HTTP/2, ya que, si enviamos el recurso sin esperar a la petición del cliente, este debería ahorrar mensajes de ida y vuelta y la latencia correspondiente, sin embargo, no responde bien ante situaciones de alta latencia.

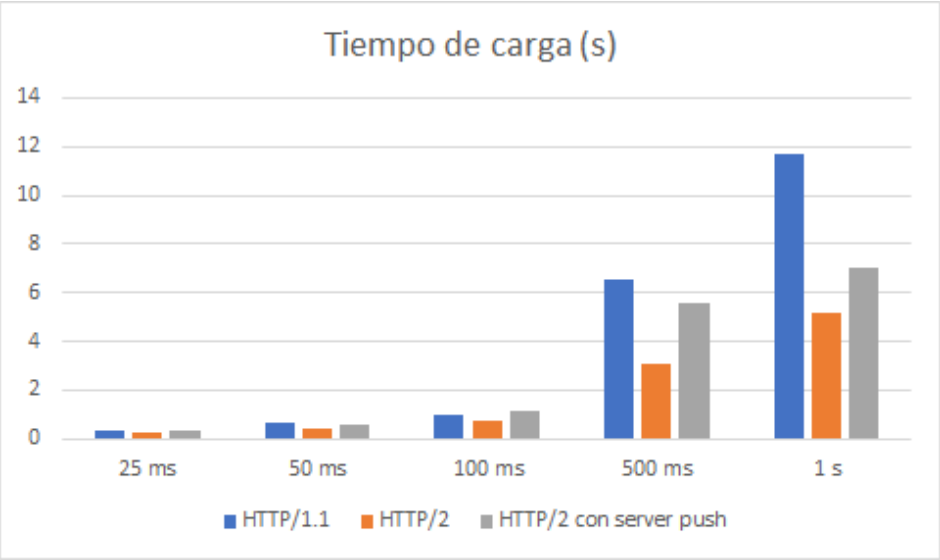

Gráfico 3. Tiempo de carga añadiendo latencia - Escenario 1

Pérdida de paquetes a velocidad de 300Mbps y sin latencia

Para este último caso del primer escenario, vamos a realizar de nuevo una petición al servidor con un ancho de banda de 300Mbps, pero esta vez, sin latencia y simulando distintos valores porcentuales de pérdida de paquetes.

Como era de esperar, ante un escenario con pocas pérdidas de paquetes, no hay un gran impacto en el tiempo de carga de la página. Para pérdidas de un 25% de los paquetes, a pesar de que con HTTP/2 obtenemos los peores resultados en cuanto al tiempo de carga, son muy similares a los otros dos casos, por lo que tampoco se puede determinar con certeza un escenario mejor al resto. Para 50%, no se ha podido determinar un tiempo aproximado debido a las grandes diferencias en los tiempos de carga. Detallaremos más este punto al final del capítulo.

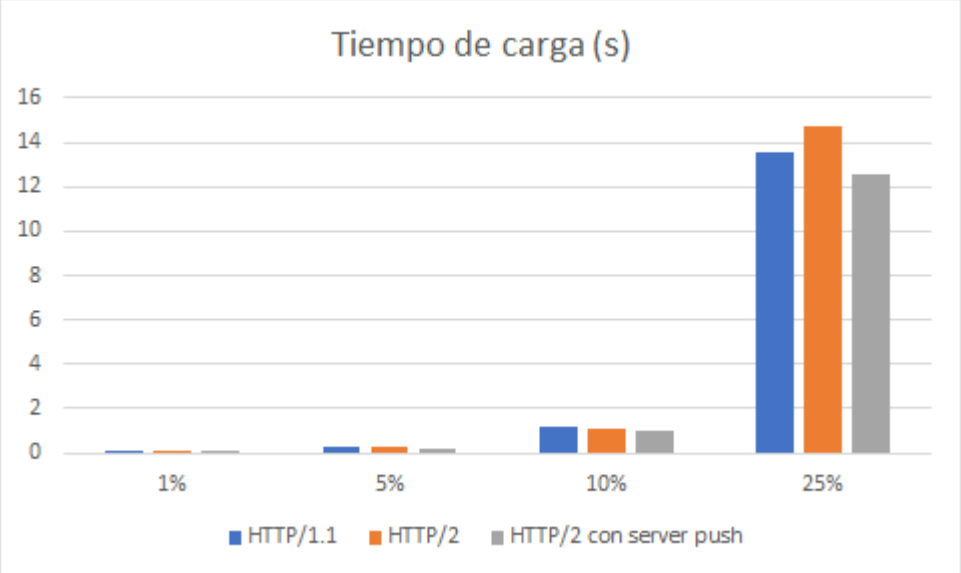

Gráfico 4. Tiempo de carga simulando pérdida de paquetes - Escenario 1

En resumen, en este primer escenario, hemos visto diferentes casos para analizar cómo se comporta HTTP/1.1, HTTP/2 y HTTP/2 con server push en la carga de una página web muy simple. Hemos determinado que tanto el ancho de banda como la pérdida de paquetes no tienen ningún impacto ante los distintos casos de estudio en los tiempos de carga. Por otro lado, el tiempo de carga mejora significativamente con el protocolo HTTP/2 para situaciones de muy alta latencia, aunque la funcionalidad server push no se comporta de la forma esperada.

4.2 Segundo escenario

Replicando los ejemplo de Akamai y de Golang que hemos mencionado anteriormente, crearemos nuestro propio ejemplo de carga de imágenes para estudiar los resultados obtenidos para peticiones con las versiones de HTTP/1.1, HTTP/2 y HTTP/2 con la funcionalidad Server Push.

Para la prueba usaremos de nuevo la imagen de 450x450px, pero esta vez dividida en 225 imágenes de 30 x 30 px y distribuida en 15 filas con 15 fragmentos cada una para la reconstrucción de la imagen original como podemos ver a continuación en el proceso de carga de la imagen.

El objetivo de esta prueba no es compararla directamente con la carga de la imagen de la prueba anterior, sino simular un escenario en el que realizamos muchas peticiones y además aprovechamos la multiplexación que nos ofrece HTTP/2.

Figura 18. Carga de imagen fragmentada con HTTP.

En cuanto a la técnica de Server Push, hemos optado por enviar todos los fragmentos para construir la imagen al mismo tiempo, sin la necesidad de que el cliente los solicite explícitamente.

Como vemos en la siguiente imagen, podemos comprobar que los recursos están siendo enviados con esta nueva funcionalidad en la herramienta de desarrolladores de los navegadores, en el apartado Initiator. Además, vemos otros datos como el protocolo que se está utilizando, el número de peticiones o el tamaño de los datos que se transfieren.

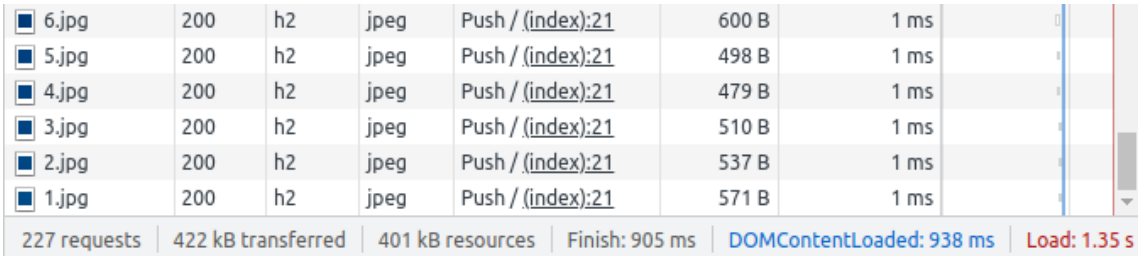

Figura 19. Funcionalidad server push en la herramienta de desarrolladores de Chrome.

Limitando ancho de banda, sin latencia y sin pérdidas de paquetes

Siguiendo el mismo procedimiento que para el escenario anterior, vamos a limitar el ancho de banda, sin añadir latencia ni pérdida de paquetes.

Para altas velocidades, podemos observar que los tres casos se comportan de manera muy similar y difieren en pocos milisegundos en todas las pruebas.

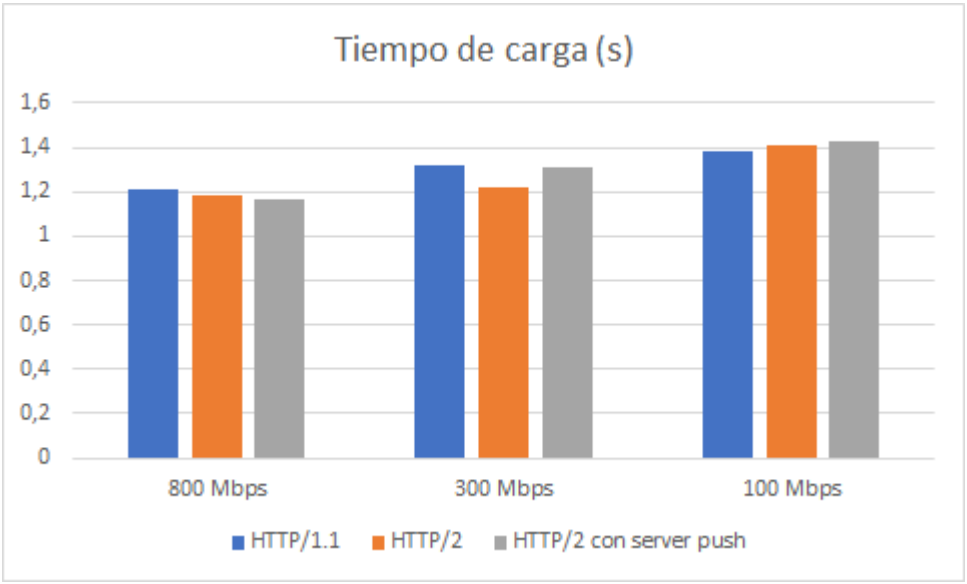

Gráfico 5. Tiempo de carga limitando ancho de banda - Escenario 2

A diferencia del escenario anterior, donde todos los casos se comportaban de manera muy similar, a medida que disminuye la velocidad, las diferencias se van haciendo más notables a favor de HTTP/2, que mejora el tiempo de carga hasta en 3s para el caso con peor ancho de banda. Por otro lado, la funcionalidad de server push parece no tener un impacto en los resultados.

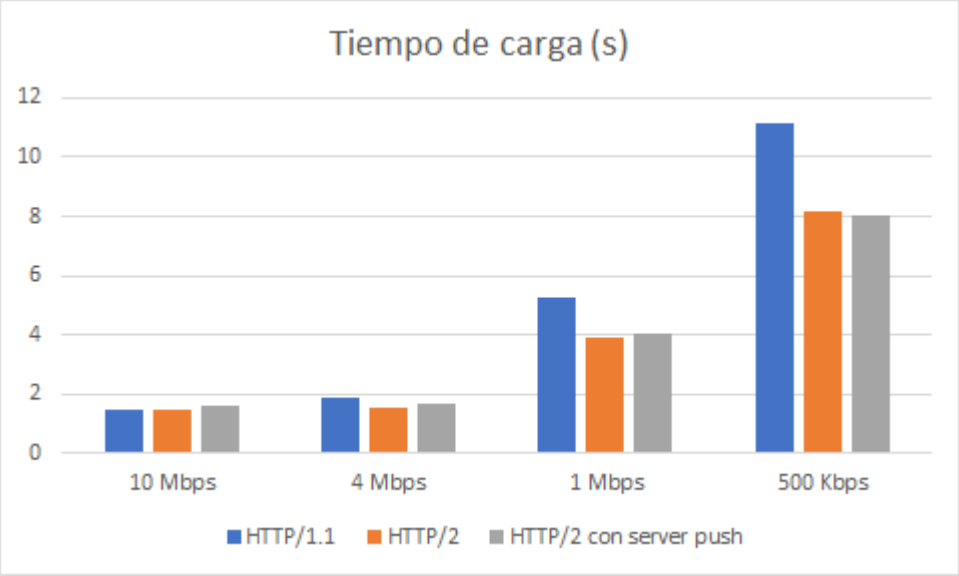

Gráfico 6. Tiempo de carga limitando ancho de banda (2) - Escenario 2

Añadiendo latencia con velocidad de 300Mbps y sin pérdida de paquetes

Para esta prueba obtenemos un resultado similar al escenario anterior, con la diferencia de que al tener que realizarse muchas más solicitudes, los tiempos de carga se agrandan mucho (nótese que la gráfica está en escala logarítmica). Por poner un ejemplo, para el caso más extremo con una latencia de 1000 ms, las peticiones con HTTP/1.1 tardan una media de 57.13 segundos, mientras que solo tardan 12.38 segundos con HTTP/2 y 12.06 con server push. Y de nuevo, no se aprecian grandes diferencias para la funcionalidad de server push.

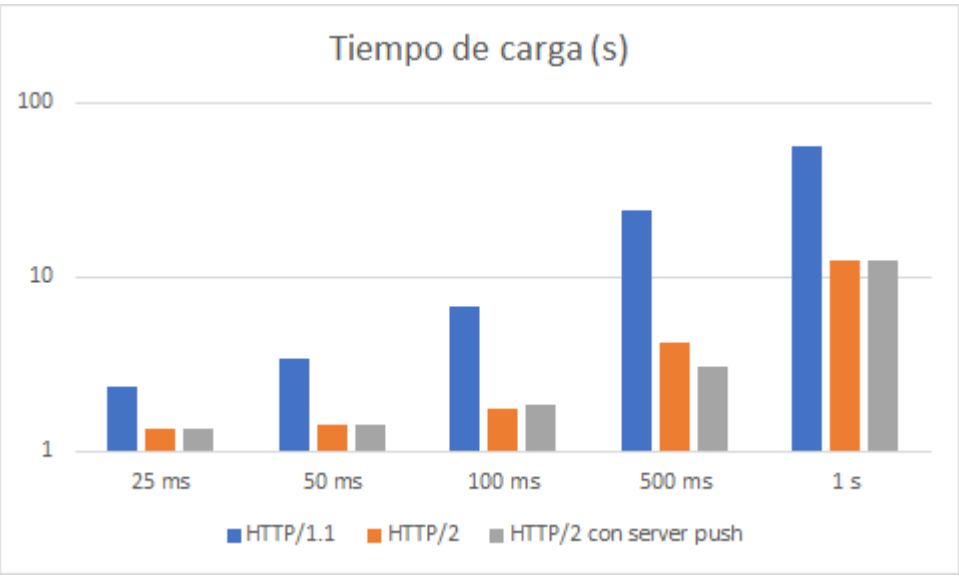

Gráfico 7. Tiempo de carga añadiendo latencia - Escenario 2

Pérdida de paquetes a velocidad de 300Mbps y sin latencia

Para el último caso de este escenario vamos a simular la pérdida de paquetes sin añadir latencia.

Como podemos ver, los tiempos de carga para pérdidas de 10% o inferiores no se ven afectadas. Para pérdidas del 25% aumenta notablemente el tiempo de carga, pero no existen diferencias entre los tres casos de estudio. Para pérdidas del 50%, no se han podido obtener resultados determinantes.

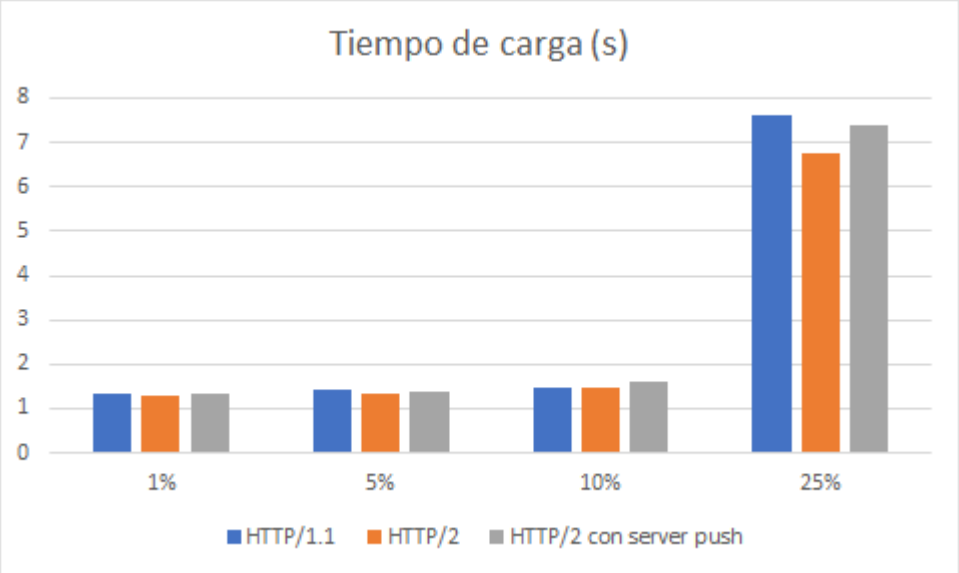

Gráfico 8. Tiempo de carga simulando pérdida de paquetes - Escenario 2

4.3 Tercer escenario

En la idea original para el trabajo teníamos pensado crear todos los escenarios en la red privada virtual, para poder realizar todas las pruebas en un mismo entorno. Aunque cuando comenzamos con el trabajo QUIC aún no había sido aprobado por la IETF, existen varias implementaciones gratuitas de HTTP/3 disponibles en Internet. Sin embargo, estas implementaciones no son servidores en sí mismas, sino que se trata de “parches” que hay que aplicar a la configuración de un servidor. Dedicamos bastante tiempo tratando de configurar estos servidores, sin embargo, no logramos que nos sirvieran tráfico HTTP/3.

Dadas las complicaciones con las que nos encontramos en la configuración del servidor HTTP/3, decidimos añadir este último escenario, donde estudiamos, siguiendo la misma filosofía que en los casos anteriores, las prestaciones de dos de los sitios web que llevan tiempo haciendo uso de la última versión del protocolo, YouTube y FaceBook. En concreto, utilizan la versión h3-29, es decir, la versión estable 29 del borrador de HTTP/3.

Para realizar las pruebas alternando entre HTTP/2 Y HTTP/3, tendremos que acceder a las funcionalidades experimentales de los navegadores, en los enlaces chrome://flags o about:config, para Chrome y Firefox respectivamente. Como podemos ver en la siguiente imagen, buscaremos la opción “Experimental QUIC protocol” y la activaremos o desactivaremos dependiendo del estudio que vayamos a realizar.

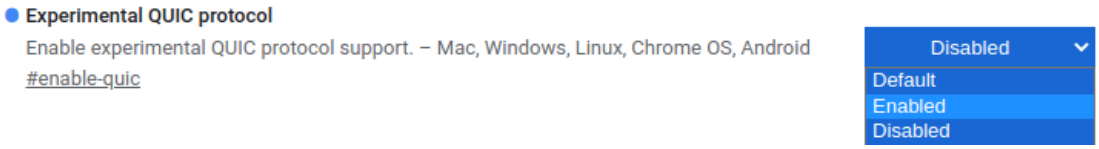

Figura 20. Activar protocolo experimental QUIC en el navegador.

YOUTUBE

Limitando ancho de banda, sin latencia y sin pérdidas de paquetes

Para la primera comparación de los protocolos HTTP/2 y HTTP/3 en un entorno real ya podemos observar algunas diferencias. Para velocidades inferiores a los 10Mbps, los tiempos de carga son mejores con el uso de HTTP/3. No se aprecian diferencias notables para situaciones con alta velocidad de conexión.

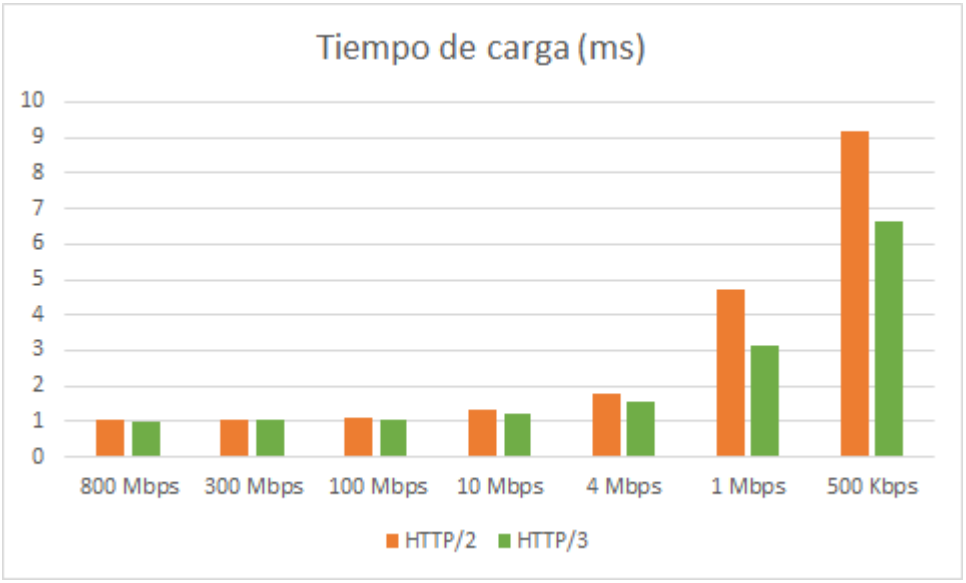

Gráfico 9. Tiempo de carga limitando ancho de banda - Youtube

Añadiendo latencia a con velocidad de 300Mbps y sin pérdida de paquetes

Del mismo modo que HTTP/2 se comportaba mejor que HTTP/1.1 para situaciones de alta latencia, HTTP/3 consigue mejorar los tiempos de carga para dichos escenarios.

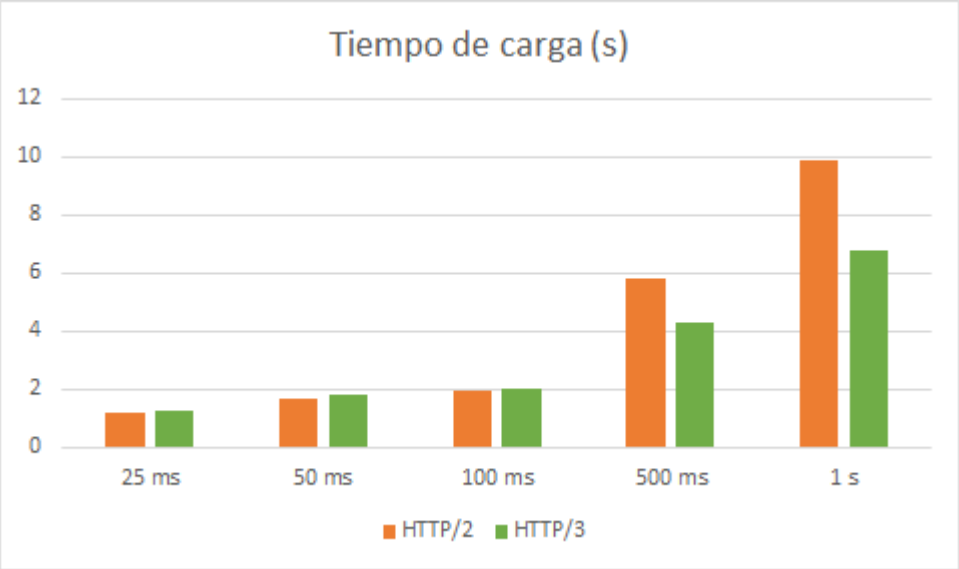

Gráfico 10. Tiempo de carga añadiendo latencia - Youtube

Pérdida de paquetes a velocidad de 300Mbps y sin latencia

En el siguiente gráfico solo mostramos tres escenarios, donde podemos ver que ambas versiones se comportan de manera similar para 1 y 5% de pérdidas de paquetes y con una pequeña diferenciación a favor de HTTP/3 para pérdidas del 10%.

En cuanto a las pérdidas del 25%, no se comportan del mismo modo en un escenario real que en uno virtual. Para las 10 peticiones que realizamos con HTTP/2 sobre el sitio web, solo obtuvimos respuesta en 3 de ellas, con tiempos de 6, 32 y 37 segundos aproximadamente. Para el resto, pasados 10 minutos, determinamos que no había respuesta, aunque hubiera cargado parte de la página. Por otro lado, con HTTP/3 obtuvimos respuesta en todas las peticiones, con un tiempo medio de 1,44 segundos.

Para las pérdidas del 50%, de las 10 peticiones con HTTP/2, obtuvimos dos respuestas, para 52 segundos y 3,3 minutos. Por su parte, HTTP/3 obtuvimos 5 respuestas con una media de unos 20 segundos.

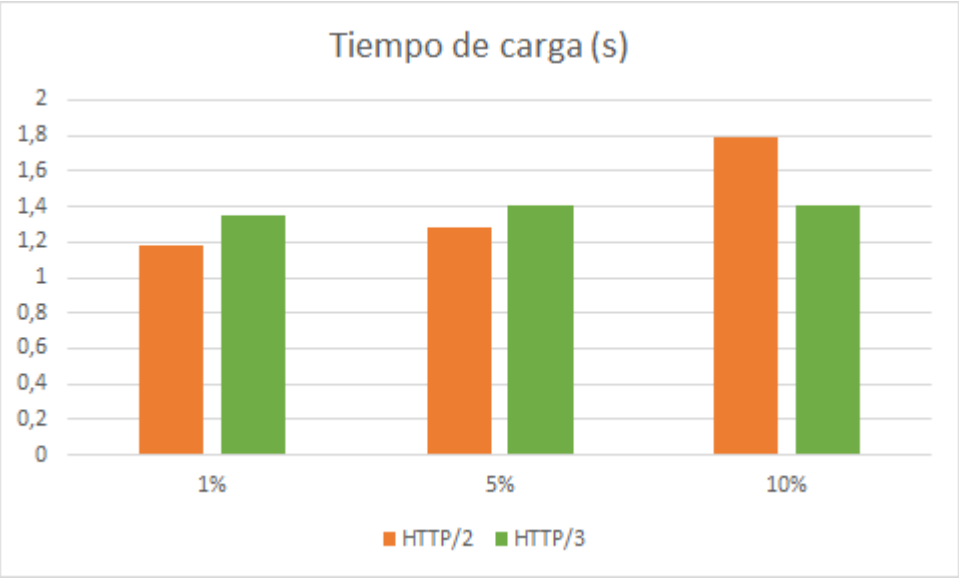

Gráfico 11. Tiempo de carga simulando pérdida de paquetes - Youtube

FACEBOOK

Limitando ancho de banda, sin latencia y sin pérdidas de paquetes

En contraposición a los resultados obtenidos en el caso anterior, en este caso, el protocolo HTTP/3 se comporta peor ante escenarios con baja velocidad de conexión. Sin embargo, se comporta de manera muy similar para altas velocidades.

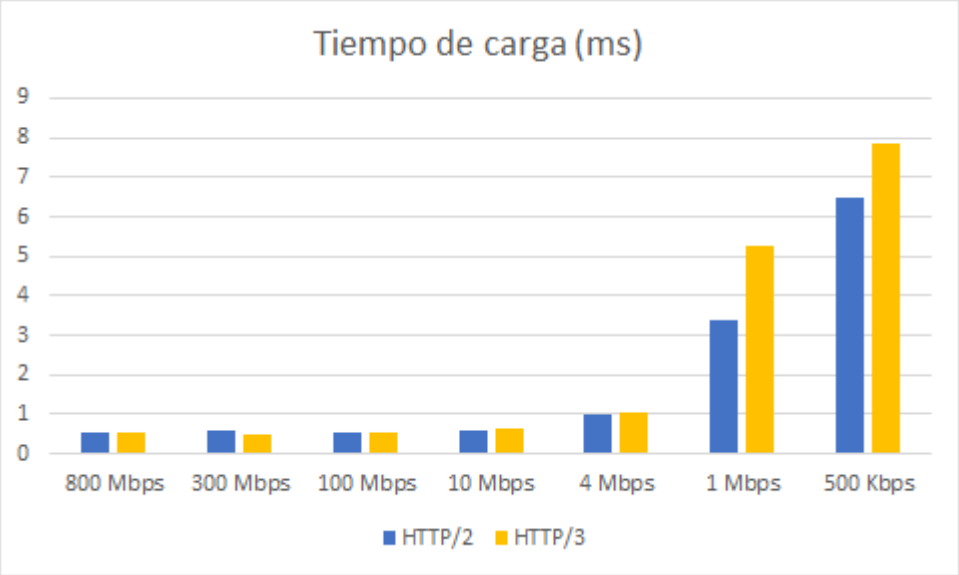

Gráfico 12. Tiempo de carga limitando ancho de banda - Facebook

Añadiendo latencia a con velocidad de 300Mbps y sin pérdida de paquetes

Para un escenario en el que añadimos latencia a la transmisión, el protocolo HTTP/3 parece comportarse mejor para ambos casos. Ya obteníamos mejoras en los tiempos de carga con HTTP/2 sobre HTTP/1.1 por lo que podemos determinar que se ha conseguido un gran avance en este aspecto.

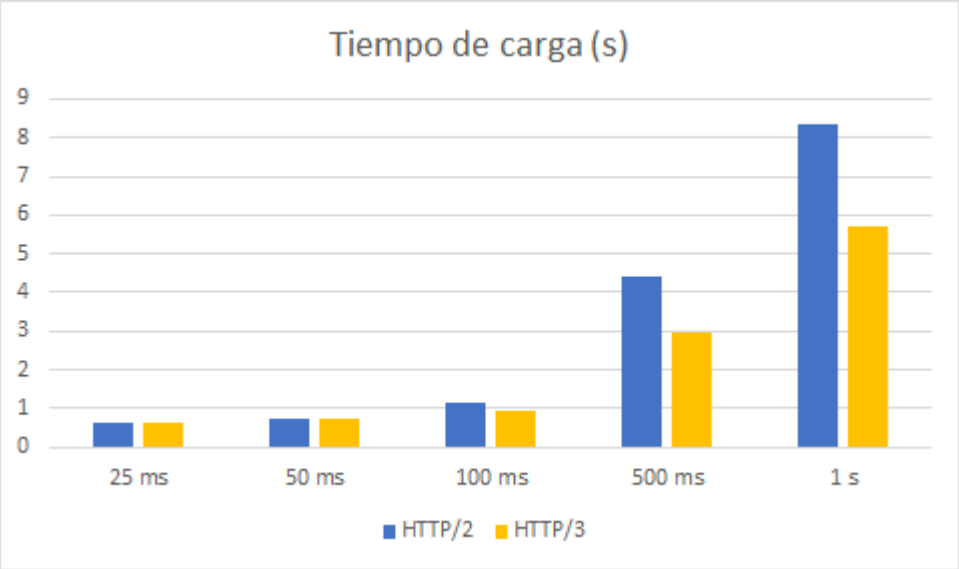

Gráfico 13. Tiempo de carga añadiendo latencia - Facebook

Pérdida de paquetes a velocidad de 300Mbps y sin latencia

En cuanto a la pérdida de paquetes nos encontramos con un escenario similar al de las pruebas sobre Youtube donde vemos que HTTP/3 se comporta mejor para pérdidas del 10%.

Para pérdidas del 25%, de las 10 solicitudes realizadas, solo obtuvimos una respuesta para HTTP/2 con aproximadamente 1.5 segundos y para pérdidas del 50% no obtuvimos ninguna respuesta.

Por otro lado, para HTTP/3, con una tasa del 25% obtuvimos todas las respuestas, con una media de 650 ms. Sin embargo, para pérdidas del 50%, solo obtuvimos 6 de las 10 peticiones que mandamos, con un tiempo medio de 3 segundos.

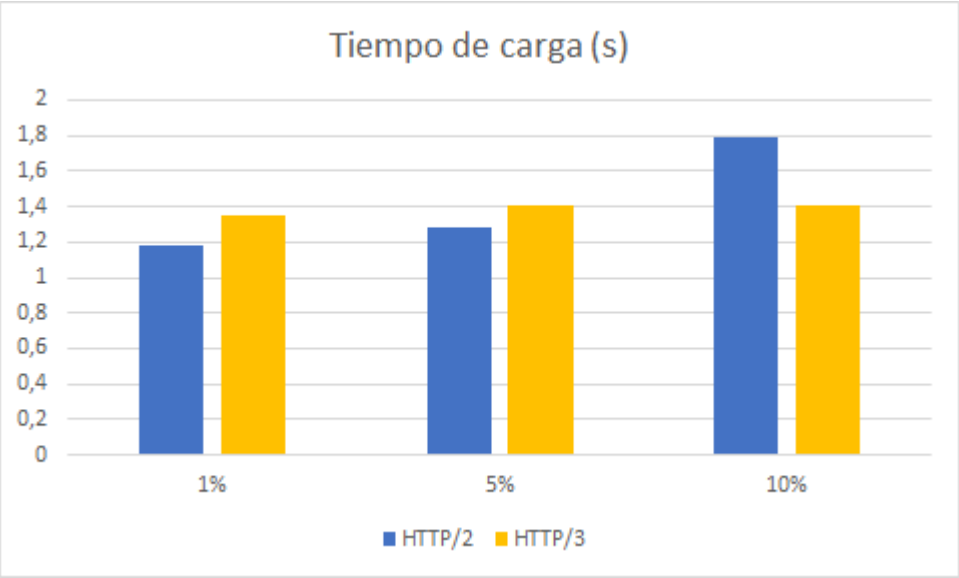

Gráfico 14. Tiempo de carga simulando pérdida de paquetes - Facebook

Para finalizar con las pruebas, quisimos comprobar por qué con una alta tasa de pérdida de paquetes existían ciertas situaciones donde obteníamos la página sin problemas en un tiempo asequible para la pérdida de paquetes que experimentamos y otras donde pasados 10 minutos no obteníamos respuesta.

Para ello, hicimos uso de WireShark, donde tuvimos que desencriptar el tráfico HTTP/2, ya que dicha funcionalidad aún no está en disponible para HTTP/3, pero se encuentra en desarrollo.

La respuesta que encontramos es que el servidor, simplemente cierra la conexión ante este tipo de situaciones al no poder trabajar con un intercambio de conexiones tan inestable. A nivel de usuario, esto se traduce en que la página se queda cargando indefinidamente, debido a que dejan de intercambiarse datos.

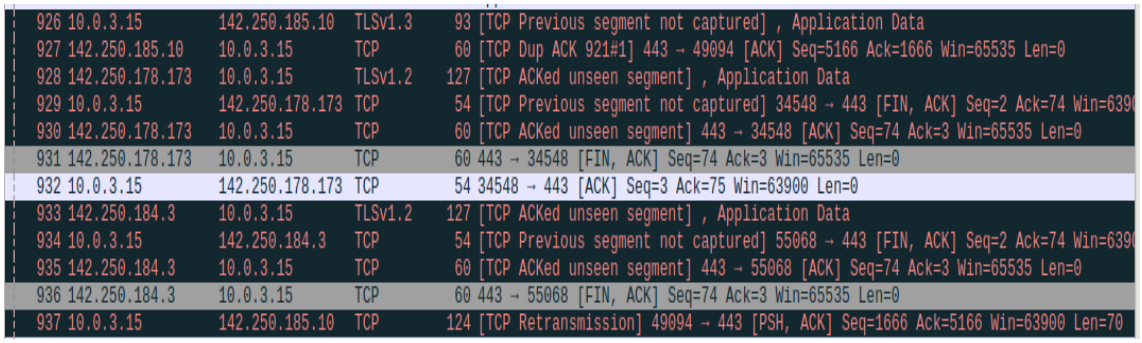

Figura 21. Cierre de conexión debido a la alta tasa de pérdida de paquetes.

5. Conclusión y líneas futuras

Hemos podido comprobar todas las ventajas que nos ofrecía HTTP/2 sobre HTTP/1.1 para adaptarse a las exigencias del Internet de hoy en día y que además, eran mejoradas por HTTP/3. Con el desarrollo de QUIC, en esta última versión se eliminan todas las desventajas ocasionadas por el protocolo de transporte subyacente que utilizaban las versiones anteriores.

En cuanto a los resultados obtenidos en los escenarios prácticos que desarrollamos podemos concluir, principalmente haciendo referencia al segundo escenario, que HTTP/2 ofrece mejores prestaciones que la versión HTTP/1.1.

Según el estudio realizado, HTTP/3 ya está muy extendido gracias a su uso por parte de las grandes empresas de Internet como Google y FaceBook, ya que mueven un tráfico muy elevado por todos sus sitios web. A pesar de que estos sitios todavía no hacen uso de la versión definitiva del protocolo HTTP/3, sino las especificaciones del borrador h3-29, según las pruebas realizadas en el tercer escenario, ofrece muy buenos resultados en las búsquedas.

Por tanto, pensamos que HTTP/2, a pesar de haber iniciado el cambio y propuesto varias funcionalidades que ahora usa HTTP/3, será reemplazado en un futuro no muy lejano por la nueva versión basada en QUIC y tal vez veamos pronto algún otro protocolo de la capa de aplicación que haga uso de él, ya que era uno de los objetivos de la IETF.

En cuanto a las líneas futuras de investigación, ahora que definitivamente ha sido estandarizado QUIC y que pronto lo hará HTTP/3, esperemos que los principales servidores como Apache y Nginx saquen a la luz un servidor con esta versión oficial del protocolo. Este escenario puede dar lugar a infinidad de pruebas sobre el mismo protocolo, como puede ser:

- Estudio de las migraciones que introducían los identificadores de conexión para comprobar la efectividad de las mismas, analizar sus ventajas y desventajas o comparar directamente con HTTP/2, que no incluye esta funcionalidad.
- Evaluación del comportamiento de HTTP/3 en redes móviles y comparación con las versiones anteriores para comprobar si también ofrece mejoras en dicho entorno.
- Estudio de la relación QUIC - TLS, principalmente en lo referente a la reanudación de la conexión de forma cifrada, para analizar qué ventajas y desventajas podría tener.
- Implementación de QUIC como base para alguno de los protocolos de la capa de aplicación como pudieran ser FTP, TFTP, SMTP, POP, DHCP, DNS...

6. Referencias

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

[2] The Chromium Projects. (2009). *SPDY: An experimental protocol for a faster web.* <https://www.chromium.org/spdy/spdy-whitepaper>

[3] Belshe, M; Thomson, M; Peon, R. (2015). *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540 <https://tools.ietf.org/html/rfc7540>

[4] Peon, R; Ruellan, H. (2015). *HPACK: Header Compression for HTTP/2.* RFC 7541 <https://datatracker.ietf.org/doc/html/rfc7541>

[5] Stenberg, D. (2014) - *HTTP2 explained* <https://http2-explained.haxx.se>

[6] Fisher, D. (2013). *CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions*. Threatpost. <https://threatpost.com/crime-attack-uses-compression-ratio-tls-requests-side-channel-hijack-secure-sessions-091312/77006/>

[7] S. Wei and V. Swaminathan. *Low latency live video streaming over http 2.0. In Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 37. ACM, 2014.

[8] The Chromium Projects. (2009). *Server Push and Server Hints* <https://www.chromium.org/spdy/link-headers-and-server-hint>

[9] Jackson, B. (2017). *HTTP/2 Statistics - KeyCDN Report on HTTP/2 Distribution* <https://www.keycdn.com/blog/http2-statistics>

[10] Langley, A. (2012). *Transport Layer Security (TLS) Next Protocol Negotiation Extension* <https://tools.ietf.org/id/draft-agl-tls-nextprotoneg-03.html>

[11] Friedl, S; Popov, A; Langley, A; Stephan, E. (2014). *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension* <https://datatracker.ietf.org/doc/html/rfc7301>

[12] Rosking, J. (2013). *Multiplexed stream transport over UDP.* <https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit>

[13] Langley, A. *et al.* (2013) *The QUIC Transport Protocol: Design and Internet-Scale Deployment* <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/8b935debf13bd176a08326738f5f88ad115a071e.pdf>

[14] The Chromium Projects. (2020). *Chrome is deploying HTTP/3 and IETF QUIC* <https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.html>

[15] Stenberg, D. (2018). *HTTP/3 explained* <https://http3-explained.haxx.se/en>

[16] Fernández, F. (2019) - *Mecanismos de control de congestión y errores en el protocolo QUIC* <https://repositorio.unican.es/xmlui/bitstream/handle/10902/17072/419984.pdf?sequence=1&isAllowed=y>

[17] Bishop, M; Krasic, C; Frindell, A.(2021) *QPACK: Header Compression for HTTP/3* <https://datatracker.ietf.org/doc/draft-ietf-quic-qpack/>

[18] Iyengar, J; Oku, K. (2020) - *Can QUIC match TCP's computational efficiency?* <https://www.fastly.com/blog/measuring-quic-vs-tcp-computational-efficiency>

[19] Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3* <https://datatracker.ietf.org/doc/html/rfc8446>

[20] Dierks, T; Rescorla, E. (2008). *The Transport Layer Security (TLS) Protocol Version 1.2* <https://datatracker.ietf.org/doc/html/rfc5246>

7. Anexos

Anexo A. Configuración de una red privada virtual con VirtualBox

VirtualBox es un programa gratuito de Oracle y se trata de un software multiplataforma, con lo que sirve tanto para Linux, Windows o Mac. En nuestro caso vamos a instalarlo en una máquina anfitrión Windows y a fecha de realización de las pruebas se va a utilizar la última versión, la 6.1.22.

Vamos a trabajar con una red virtual que conectará de forma privada la máquina anfitriona (en la que estas instalando el VirtualBox) y las máquinas virtuales que creemos en ella. Para crear esta red debemos acceder al menú “Archivo → Administración de red de anfitrión”. Por lo general, ya viene con una red creada por defecto con el nombre “Virtualbox Host-Only Ethernet Adapter” (para SO Windows).

A continuación, debemos activar y configurar el servidor DHCP de esta red. La configuración que hemos utilizado para el servidor se puede apreciar en la siguiente imagen.

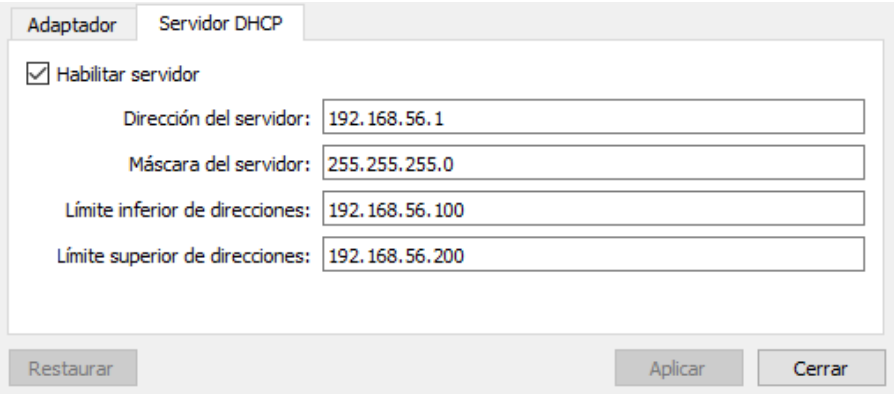

Figura 19. Configuración Servidor DHCP en red privada virtual.

En cuanto a las máquinas que vamos a utilizar, tenemos un sistema Linux con distribución Ubuntu 18.04 LTS (bionic). Para contar con una máquina que actúe como servidor y otra como cliente, vamos a clonarla, pulsando con el botón derecho sobre dicha máquina virtual y seleccionando la opción “Clonar...”. A la nueva máquina la llamaremos “Cliente” y en la política de dirección MAC, escoger la opción “Generar nuevas direcciones MAC para todos los adaptadores de red”, como podemos ver en la siguiente imagen.

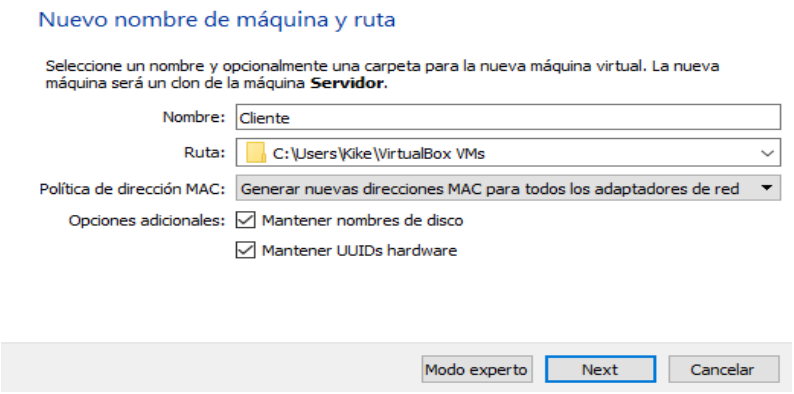

Figura 22. Clonación máquina virtual “Cliente” para la red privada.

Una vez tenemos creadas las dos máquinas, “Servidor” y “Cliente”, vamos a configurar la red de ambas máquinas. Para ello, accedemos al apartado “Configuración → Red”.

En el caso del servidor, vamos a configurar dos tarjetas de red, ya que el servidor estará conectado a la red interna que creamos antes mediante una tarjeta de red, y al router virtual que le da acceso a Internet mediante otra tarjeta de red. En el apartado “Red”, el adaptador 1 lo emplearemos para el acceso a la red privada, y debe quedar configurado como en la siguiente figura (el nombre de la red será el que apareció al configurar la red sólo anfitrión).

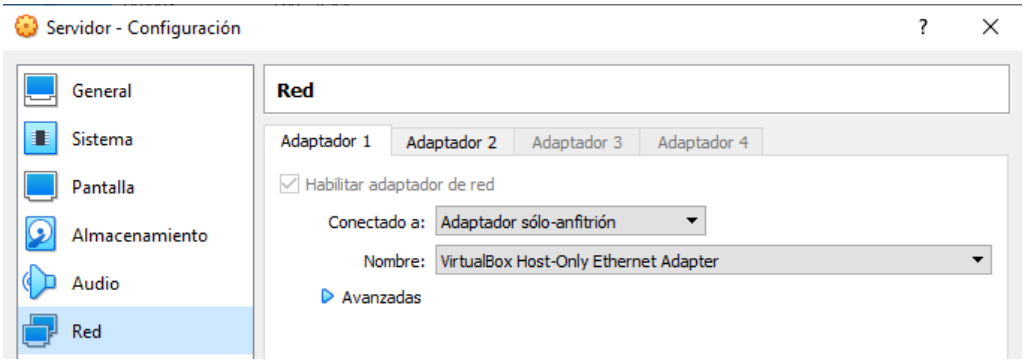

Figura 23. Configuración adaptador 1 para máquinas “Servidor” y “Cliente”.

El adaptador 2 debe quedar configurado como en la siguiente figura.

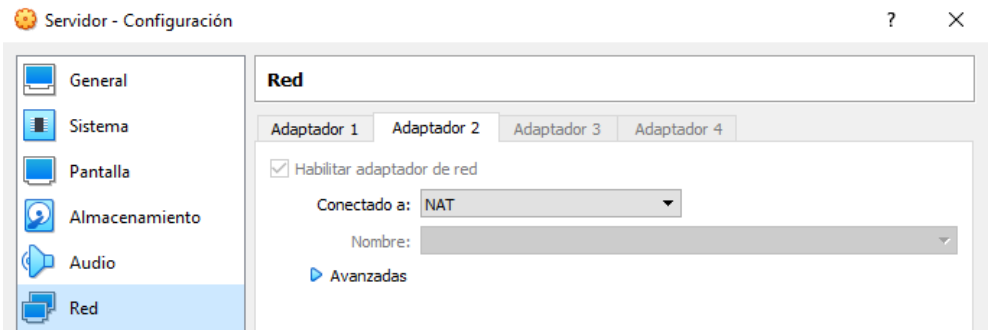

Figura 24. Configuración adaptador 2 para máquina “Servidor”.

Ahora vamos a configurar la red de la máquina virtual “Cliente”. Esta máquina virtual sólo tiene una interfaz de red conectada al switch virtual, ya que no se va a conectar a Internet, sino que solo tendrá acceso al servidor web que instalemos en la máquina “Servidor”. En este caso debemos de configurar el “Adaptador 1” de la misma forma que en el caso anterior (como “adaptador sólo-anfitrión”).

Con el modo solo anfitrión podremos conectarnos desde el anfitrión a nuestras máquinas virtuales y viceversa, así como conectarnos entre las máquinas virtuales, es decir, dentro de la red privada. En todo caso, no tendremos por defecto conexión a Internet.

El modo NAT sería todo lo contrario al modo anterior. Las máquinas virtuales tendrían salida a Internet, pero no podríamos conectarnos entre diferentes máquinas virtuales.

En definitiva, la configuración final con la que contaremos para la realización de las pruebas se puede apreciar en la siguiente figura.

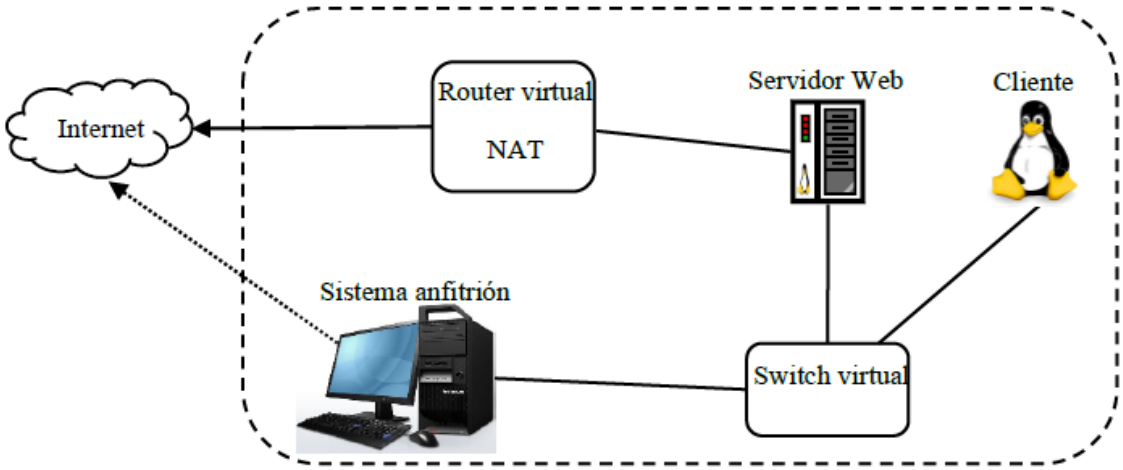

Figura 25. Esquema de la configuración final para la red privada virtual.

Anexo B. Instalación de servidor con servicio HTTP/2 en Apache

Instalación del servidor Apache y los paquetes necesarios en la máquina virtual “Servidor”.

```
apt-get update
apt-get install apache2 openssl
```

Habilitar los módulos ssl y rewrite del servidor.

```
a2enmod ssl
a2enmod rewrite
```

Editar el archivo de configuración de Apache /etc/apache2/apache2.conf y añadir las siguientes líneas al final del archivo.

```
<Directory /var/www/html>
AllowOverride All
</Directory>
```

Como vamos a trabajar con HTTP/2, es necesario instalar un certificado, en nuestro caso es autofirmado. Para ello, utilizaremos la herramienta OpenSSL.

```
mkdir /etc/apache2/certificate
cd /etc/apache2/certificate
openssl req -new -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out apache-certificate.crt -keyout apache.key
```

Ahora debemos editar el archivo de configuración para nuestro sitio web, que se encuentra disponible en /etc/apache2/sites-enabled/000-default.conf y añadimos el certificado al puerto 443

```
<VirtualHost *:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile /etc/apache2/certificate/apache-certificate.crt
        SSLCertificateKeyFile /etc/apache2/certificate/apache.key
</VirtualHost>
```

Para acabar con esta parte reiniciamos el servicio Apache.

```
service apache2 restart
```

Ahora nuestro servidor está disponible para las máquinas presentes en la red, pero solo podemos acceder a él mediante HTTP (puerto 80) y HTTPS (puerto 443). Los módulos necesarios para HTTP/2 vienen por defecto con el paquete de instalación a partir de la versión 2.4.17 de Apache, pero debemos habilitarlos manualmente.

A continuación, vamos a habilitar la funcionalidad para HTTP/2.

a2enmod http2

Y editamos de nuevo el archivo de configuración, que podemos encontrar en /etc/apache2/sites-enabled/000-default.conf para añadir las siguientes instrucciones justo después de la directiva <VirtualHost *: 443>, del siguiente modo:

<VirtualHost *: 443>
LoadModule http2_module modules/mod_http2.so
Protocols h2 http / 1.1
……..
</VirtualHost>

Por último, reiniciamos el servicio Apache

service apache2 restart

Ahora nuestro sitio web tiene habilitada la funcionalidad HTTP/2.

Como también pretendemos hacer uso de la funcionalidad de server push de HTTP/2, podremos activarla/desactivarla con la directiva:

H2Push On/Off

Para hacer “push” de los recursos podremos utilizar las siguientes directivas

H2PushResource /xxx.css
H2PushResource /xxx.js
H2PushResource /xxx.jpg