UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Correlación de trazas de red con registros de servidor para monitorizar la calidad de servicio de conexiones cifradas

Máster Universitario en Ingeniería de Telecomunicación

Autor: González Hernández, David

Tutor: López de Vergara Méndez, Jorge E.

Departamento de Tecnología Electrónica y de las Comunicaciones

FECHA: Septiembre, 2020

Título: Correlación de trazas de red con registros de servidor para monitorizar la calidad de servicio de conexiones cifradas

Autor: González Hernández, David

Tutor: López de Vergara Méndez, Jorge E.

High Performance Computing and Networking Research Group

Departamento de Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Madrid, septiembre 2020

Resumen

Este Trabajo de Fin de Máster se centra en la monitorización de los servidores HTTP y HTTPS. Dado que en la actualidad el uso de Internet está creciendo y la carga de los servidores va en aumento, monitorizar el estado de los servidores se está cada vez convirtiendo en una tarea más importante de cara a poder detectar, anticipar y analizar los posibles problemas que se produzcan.

En cuanto a la monitorización sobre servidores HTTP, se han realizado gran cantidad de estudios y metodologías para medir la efectividad del rendimiento de los servidores en los que se analizan los tiempos de respuesta a partir del tráfico de red. Pero ahora que el uso del protocolo HTTPS está aumentando, es necesario medir el rendimiento de este tipo de servicio. Debido a que este protocolo cifra los mensaje que se intercambian el cliente y el servidor, sus tiempos de respuesta ya no son tan fácilmente medibles como en HTTP.

Por lo tanto, para resolver esta situación, primero se plantea el desarrollo de un algoritmo que correle los *logs* de acceso del servidor HTTP con la captura de tráfico de red. Posteriormente se adaptará dicho algoritmo para que se pueda aplicar sobre servidores HTTPS. Con ellos, seremos capaces de identificar dónde se ha producido el fallo y poder mejorar la calidad de servicio prestada a los clientes.

Palabras Clave

Monitorización de tráfico, *log* de acceso, tráfico HTTP, tráfico HTTPS, calidad de servicio, servidores web, función de distribución acumulada, retardos, pérdida de paquetes, tiempo de respuesta

Abstract

This Master's Thesis is focused on the monitorization of HTTP and HTTPS servers. Nowadays, the use of Internet is currently raising and the load on servers is increasing as well. Thus, monitoring the status of the servers is becoming an important task to detect, anticipate and analyze possible problems that may occur.

Regarding the monitorization of HTTP servers, many studies and methodologies have been carried out to measure the effectiveness of the performance of the servers in which response times are analyzed based on the network traffic. But now that the use of HTTPS is increasing, it is necessary to measure the performance of these types of services. Because this protocol encrypts messages sent between client and server, its response times are no longer as easily measurable as in HTTP server.

Therefore, to solve this situation, the development of an algorithm that correlates the HTTP server access *logs* with the capture of network traffic is proposed. This algorithm will be adapted, and it can be applied on HTTPS servers. With these methods, we will be able to identify where the failure has occurred and to improve the quality of service provided to the customers.

Keywords

Traffic monitoring, Access *log*, unencrypted traffic, encrypted traffic, quality of service, web server, cumulative distribution function, delay, packet loss, response time

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutor Jorge por haberme dado la oportunidad de realizar este proyecto y porque, aún compaginando el Máster con el trabajo a jornada completa, siempre ha estado pendiente, apoyándome y animándome a realizarlo. Gracias por todo.

Agradecer a mi madre todo lo que hizo por mí y porque siempre fue un apoyo esencial durante mi vida, sé que estaría orgullosa de que haya podido finalizar el Máster; y a mi padre, que sin él no hubiese sido capaz de seguir adelante y estar donde estoy. Os quiero.

Por último, quisiera dar las gracias a Valeria por estar tanto en los momentos buenos como en los no tan buenos y por su apoyo incondicional, siempre ha estado animándome a seguir adelante y a no abandonar.

Índice General

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Motivación

La dependencia actual que tenemos con Internet y la gran cantidad de aplicaciones web que existen, se ha producido un aumento drástico en la carga de los servidores, por lo que monitorizar el estado del servidor y su rendimiento es una tarea bastante importante para detectar y anticipar los posibles problemas que puedan surgir (retardos, cuellos de botella, pérdida de paquetes…). Debido a estas causas, el cliente final no obtiene la calidad de servicio deseada.

En los últimos años se han planteado diversas metodologías para estimar la efectividad del rendimiento de los servidores web [1], [2], [3] y [4] en los que se analizan los registros o *logs* de acceso del servidor y se mide el tiempo de respuesta percibido por los usuarios.

Cuando estos *logs* de acceso no son fácilmente accesibles o no proveen la información necesaria, el análisis del servicio es realizado basado en la monitorización pasiva del tráfico de red. Sin embargo, se está produciendo un incremento considerable del uso del protocolo HTTPS. La investigación sobre el análisis de tráfico HTTP es abundante; no obstante, HTTPS limita su efectividad debido al HTTPS de la carga útil. En este caso, las peticiones y respuestas HTTP están ocultas, lo cual plantea un problema en la monitorización del rendimiento de la red, ya que los tiempos de petición-respuesta ya no pueden ser medibles a partir de una captura del tráfico. Sin embargo, el HTTPS de dichas peticiones requiere acceso a la clave privada del servidor, que no puede ser accesible en todos los escenarios. Además, con la última versión (1.3) del protocolo *Transport Layer Security* (TLS) el descifrado HTTPS no es viable incluso en posesión de la clave privada del servidor. Esto es debido a que se generan claves efímeras para cada conexión realizada. A pesar de ello, los *logs* de acceso del servidor pueden contener el tiempo de respuesta de cada petición.

Con esta motivación, se propone en este Trabajo de Fin de Máster una nueva metodología en el que se implementará un algoritmo que correle el *log* de acceso del servidor con el tráfico capturado mediante herramientas de monitorización de actividad de red. Con ello se busca que este algoritmo funcione tanto para el protocolo HTTP [5] como HTTPS [6] y [7], y seamos capaces de poder identificar en qué nivel de la torre de protocolos se produce el posible fallo y poder así mejorar la calidad de servicio prestada.

1.2. Objetivos

El principal objetivo de este Trabajo de Fin de Máster es correlar los registros de acceso de dos servidores, un servidor HTTP y otro HTTPS, con las trazas de red capturadas mediante herramientas de monitorización. Para ello, se generarán diversas bases de datos que contendrán los *logs* de accesos de los servidores correspondientes y su captura equivalente de trazas de red. Posteriormente, se diseñará y aplicará un algoritmo sobre el

servidor HTTP, que será capaz de correlar el *log* de acceso con la traza capturada y podremos observar si los problemas encontrados son debidos al servidor o debido al cliente. Este mismo algoritmo se modificará para que se pueda aplicar también sobre servidores HTTPS.

Para poder cumplir con este objetivo global marcado, se ha establecido una serie de objetivos parciales para garantizar la correcta implementación de la correlación:

- Estudio de los procesos de monitorización sobre servidores mediante la medición de los tiempos de respuesta.
- Desarrollo de ambos algoritmos para correlar el *log* de acceso del servidor con la captura de tráfico de red.
- Estudio de los resultados del algoritmo de correlación, se compararán mediante métodos estadísticos y gráficamente.

El cumplimiento de estos objetivos parciales nos hará cumplir con el objetivo general de monitorizar y correlar los servidores web HTTP y HTTPS.

1.3. Plan de Desarrollo

El desarrollo de este proyecto se ha organizado en las siguientes fases:

- Fase 1: Estudio y Análisis del Problema
    - Evaluación sobre los posibles tipos de monitorización sobre la red.
    - Análisis de servidores web y herramientas de monitorización.
    - Estudio de trabajos previos relacionados con la monitorización sobre redes HTTP y HTTPS.
- Fase 2: Diseño
    - Creación y configuración de dos servidores web: uno HTTP y el otro servidor HTTPS.
    - Configuración de las herramientas de monitorización.
- Fase 3: Desarrollo
    - Generación de base de datos de trazas de red y monitorización de ambos servidores.
    - Desarrollo de un algoritmo para correlar registros y trazas de tráfico web HTTP.
    - Adaptación de dicho algoritmo para realizar la correlación sobre el servidor HTTPS.
- Fase 4: Verificación y Validación
    - Generación de las pruebas de ambos algoritmos sobre ambos servidores.
    - Análisis de los resultados.
    - Comprobación de la calidad de servicio (QoS) percibida.
    - Monitorización de ambos servidores a través de las herramientas ElasticSearch y Kibana.
- Fase 5: Documentación

- Redacción del informe del trabajo, en él se refleja el conocimiento y la experiencia adquiridos durante la realización de este Trabajo de Fin de Máster.

Este proyecto se ha realizado en una duración total de 12 meses y se ha dividido según las fases descritas anteriormente. A continuación, se muestra el diagrama de Gantt que ha seguido el Trabajo de Fin de Máster:

Fases	Horas	sep-19	oct-19	nov-19	dic-19	ene-19	feb-19	mar-19	abr-19	may-19	jun-19	jul-19	ago-19
T1. Estudio y Análisis del Problema	**40**												
T1.1 Evaluación sobre los posibles tipos de monitorización sobre la red.	10												
T1.2 Análisis de servidores web y herramientas de monitorización.	10												
T1.3 Estudio de trabajos previos relacionados con la monitorización sobre redes sin cifrar y cifradas.	20												
T2. Diseño	**20**												
T2.1 Creación y configuración de dos servidores web. uno cifrado y el otro servidor sin cifrar.	10												
T2.2 Configuración de las herramientas de monitorización.	10												
T3. Desarrollo	**130**												
T3.1 Generación de base de datos de trazas de red y monitorización de ambos servidores.	10												
T3.2 Desarrollo de un algoritmo para correlar registros y trazas de tráfico web sin cifrar.	70												
T3.3 Adaptación de dicho algoritmo para realizar la correlación sobre el servidor cifrado.	50												
T4. Verificación y Validación	**75**												
T4.1 Generación de las pruebas de ambos algoritmos sobre ambos servidores.	30												
T4.2 Análisis de los resultados.	20												
T4.3 Comprobación de la calidad de servicio (QoS) percibida.	10												
T4.4 Monitorización de ambos servidores a través de las herramientas ElasticSearch y Kibana.	15												
T5. Documentación	**35**												

Figura 1-1: Diagrama de Gantt del Proyecto

1.4. Organización de la Memoria

En esta memoria quedan reflejados los estudios realizados e implementaciones desarrolladas a lo largo del ciclo de vida de este proyecto para poder llegar a obtener conclusiones acerca de la correlación de trazas de registros.

A continuación, se muestran los capítulos en los que se organiza esta memoria:

- Capítulo 1. Introducción. Este primer capítulo consta de la motivación con la que se ha llevado a cabo este proyecto; se presentan los objetivos principales para la realización de este Trabajo de Fin de Máster; el plan de desarrollo en el que se incluye el diagrama de Gantt del proyecto; y, por último, la organización de la memoria.
- Capítulo 2. Estado del Arte. Se analiza el estado del arte relativo a estudios previos relacionados con la monitorización de los protocolos HTTP y HTTPS, así como los principales protocolos afectados y los métodos estadísticos usados para comparar los registros de red.
- Capítulo 3. Análisis del Problema. Se plantean los problemas de monitorización de trazas de red relativos a los *logs* de acceso de los servidores y la captura del tráfico mediante herramientas de monitorización.
- Capítulo 4. Diseño del Sistema. En este capítulo se refleja la configuración de los servidores web, así como de las herramientas de monitorización para filtrar los registros con la información que nos interesa.
- Capítulo 5. Desarrollo de la Solución. Se mostrará cómo se han generado las bases de datos y qué información contienen; se aplicarán los algoritmos desarrollados para ambos servidores; y, por último, se explicará cómo se ha llevado a cabo la monitorización de la red.
- Capítulo 6. Validación y Resultados. Se mostrarán y analizarán los resultados obtenidos una vez aplicados ambos algoritmos, la comparativa entre los *logs* de acceso al servidor y la captura de red se realizará mediante métodos estadísticos. También se evaluará cómo es la calidad de servicio obtenido, y, finalmente, se mostrarán gráficamente dichos resultados.
- Capítulo 7. Conclusiones y Trabajo Futuro. En este último capítulo se muestran las conclusiones más relevantes, las aportaciones que han surgido durante el desarrollo del proyecto y se propondrán propuestas para trabajos futuros.

2. Estado del Arte

2.1. Introducción

En este capítulo se detalla el estado del arte relacionado con el proyecto que se presenta. En esta primera sección se introducen los fundamentos teóricos básicos que serán necesarios para la comprensión y realización de este proyecto. En primer lugar, nos centraremos en los protocolos de red TCP, HTTP y HTTPS, explicando sus principales características. Cabe destacar la importancia y el crecimiento que está tomando el tráfico basado en HTTPS, ya que la mayoría del tráfico web se basa en este protocolo [8]. A continuación, se detallan las funciones de distribución que se aplicará sobre el tráfico web. Por último, se realiza un estudio exhaustivo sobre los servidores web y las herramientas de monitorización de red.

2.2. Protocolos TCP, HTTP y HTTPS

2.2.1. Protocolo TCP

El protocolo TCP (*Transmission Control Protocol*) [9] es unos de los protocolos fundamentales de Internet, ya que las aplicaciones pueden comunicarse entre sí de una manera altamente segura independientemente de los niveles inferiores. TCP es un protocolo orientado a conexión, es decir, que su función principal es permitir que dos máquinas que se están comunicando controlen el estado de la transmisión. Existe otro protocolo de transporte más sencillo denominado UDP (*User Datagram Protocol*) [10], el cual envía paquetes (datagramas) de un extremo a otro sin garantizar su correcta llegada, y también existe el protocolo QUIC (*Quick UDP Internet Connections*) [11] basado en UDP+TLS, es todavía un protocolo experimental y su principal objetivo es mejorar el rendimiento que ofrece TCP reduciendo su latencia y las pérdidas producidas. Cuando se muestre la efectividad de este protocolo se migrará a una versión posterior de TCP y TLS. Pero por ahora, si la aplicación necesita que la conexión sea fiable, se recurre al protocolo TCP.

Las principales características de este protocolo son las siguientes:

- Flujo de datos fiable, es decir, garantiza la recepción mutua de datos.
- Control de flujo para evitar congestionar al receptor y a la red.
- Comunicación orientada a conexión.

A continuación, se analizarán estas características:

Flujo de datos fiable

El protocolo TCP está diseñado de tal manera que se puedan recuperar paquetes debido a pérdidas, corrupciones o que los paquetes han llegado al receptor de manera desordenada. Para ello, se utilizan mecanismos de retransmisión y de reconocimiento de paquetes.

Cuando se transmiten paquetes, el protocolo TCP realiza una copia de cada uno y los coloca en una cola destinada a la retransmisión y se inicia un temporizador. Estos paquetes se descartan de la cola una vez se ha recibido el ACK correspondiente, mientras que, si este ACK no se recibe antes de que termine el temporizador, el paquete se retransmite.

Cada paquete tiene asignado un número de secuencia, por lo que, a la hora de recibir paquetes, estos números son comprobados para ordenar adecuadamente cada paquete. Con ello evitamos los duplicados y se garantiza la correcta llegada de paquetes en el orden adecuado.

Existen diversos algoritmos de retransmisión que están implementados en el protocolo TCP, como, por ejemplo: Algoritmo de Karn [12] o algoritmo de retransmisión adaptativo.

Control de flujo

El protocolo TCP es capaz de controlar la tasa de envío de datos para evitar sobrecargar tanto la red como al receptor. Este tipo de control se realiza mediante ventanas deslizantes para así poder aprovechar mejor el ancho de banda de la red transmitiendo únicamente un número concreto de paquetes antes de que llegue el ACK correspondiente.

Orientado a conexión

Para proporcionar la fiabilidad de la que se caracteriza el protocolo TCP, tiene que comprobar continuamente la información sobre el estado del flujo de datos. Cada conexión se identifica a los dos extremos de manera única.

El protocolo TCP inicia la conexión intercambiando paquetes de sincronismo (SYN) en tres pasos. A este proceso se le denomina *three way handshake*. El cliente comienza enviando el paquete SYN, el receptor lo recibe y envía un paquete SYN junto con el ACK correspondiente. Por último, el cliente envía su paquete ACK confirmando que lo ha recibido.

En este proceso no solamente se ha iniciado la conexión, sino que también se ha acordado un número de secuencia aleatorio que servirá para identificar los paquetes de la conexión. De esta manera, los dos extremos han acordado su conexión y se puede proceder al intercambio de datos.

Para finalizar la conexión, el cliente inicia este proceso enviado un paquete de finalización (FIN), la parte receptora lo recibe y envía el ACK correspondiente junto con otro paquete de finalización, el cliente envía su ACK y con esto quedaría cerrada la conexión.

A continuación, se muestra una gráfica de todo este proceso que establece el protocolo TCP:

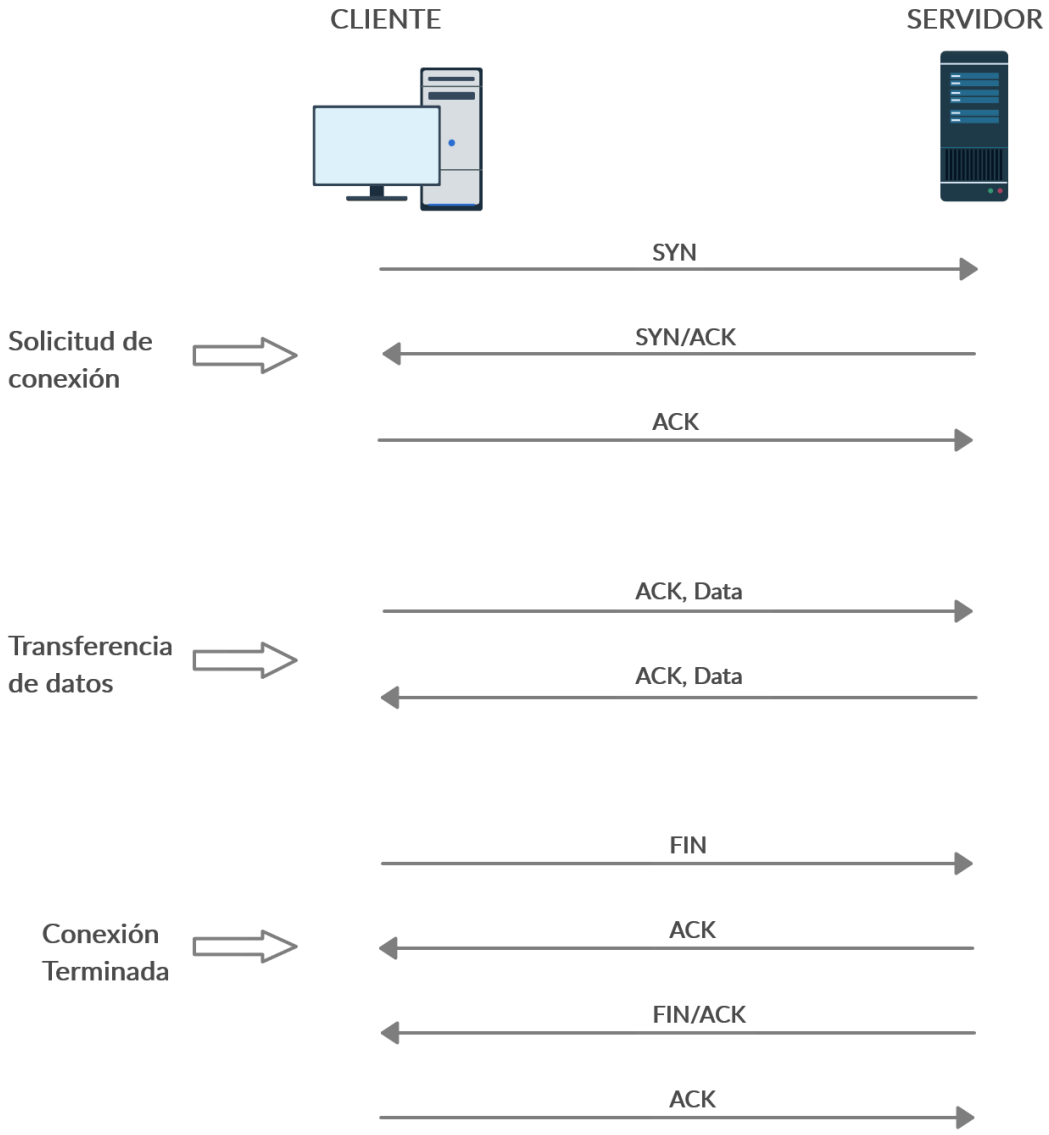

Figura 2-1: Diagrama de flujo TCP

2.2.2. Protocolo HTTP

HTTP (*HyperText Transfer Protocol*) [5] es el protocolo de transmisión entre cliente y servidor que permite el intercambio de información (HTML, CSS, imágenes, javascript…) entre ellos. Desde la perspectiva de las telecomunicaciones, este protocolo está soportado sobre conexiones del protocolo TCP en el que el servidor escucha a través del puerto 80 y espera las peticiones de los clientes. Una vez que se ha establecido la conexión entre el servidor y el cliente, el protocolo TCP se encarga de mantener segura y activa la comunicación y el intercambio de datos.

HTTP sigue un esquema basado en operaciones de petición y respuesta, las peticiones son enviadas mayormente por el cliente, que suele ser un navegador web, aunque también puede ser una aplicación que genere peticiones, y cada petición es procesada por el

servidor, el cual responde con un mensaje conteniendo el estado de la operación y el resultado.

Los recursos que el cliente indica en la petición están clasificados por la descripción MIME, de esta manera HTTP se puede encargar de intercambiar cualquier tipo de dato.

Las principales características del protocolo HTTP son las siguientes:

- Permite la transferencia de ficheros multimedia.
- Hay tres tipos básicos de peticiones que el cliente realiza:
  - GET: para recoger cualquier tipo de información del servidor.
  - POST: para enviar información al servidor.
  - HEAD: para solicitar información acerca de un objeto contenido en el servidor, por ejemplo, la última fecha de modificación de un archivo.
- Cada operación implica una conexión con el servidor y es liberada al terminar la misma. Actualmente esto se ha mejorado y la conexión se mantiene activa mediante el mecanismo *HTTP Keep Alive*.
- El servidor trata de manera independiente cada petición.

Respecto a las transacciones realizadas por el servidor, el código que devuelve a cada petición contiene información acerca del resultado de la operación. Los códigos se clasifican en cinco categorías:

- 1xx: mensajes de tipo informativos.
- 2xx: mensajes asociados a peticiones correctas.
- 3xx: mensajes de redireccionamiento.
- 4xx: mensajes de error por parte del cliente.
- 5xx: mensajes de error por parte del servidor.

La lista de códigos más comúnmente encontrados es la siguiente:

Tabla 2-1: Lista de códigos HTTP más comunes

Código	Descripción
200 OK	Operación realizada correctamente
301 Moved Permanently	El objeto al que se accede se ha movido a otro lugar permanentemente, además el servidor proporciona la nueva URL.
302 Moved Temporarily	Los datos solicitados están disponibles temporalmente en otra ubicación.

Código	Descripción
400 Bad Request	La petición del cliente tiene un error de sintaxis y el servidor no es capaz de procesarla.
403 Forbidden	El objeto solicitado está protegido y se ha denegado el acceso.
404 Not Found	La URL solicitada no existe.
500 Internal Server Error	El servidor ha tenido un error interno y no puede continuar con el procesamiento de peticiones.
503 Service Unavailable	El servidor está sobrecargado y no es capaz de gestionar las peticiones.

A continuación, se muestra un ejemplo de flujo de datos HTTP:

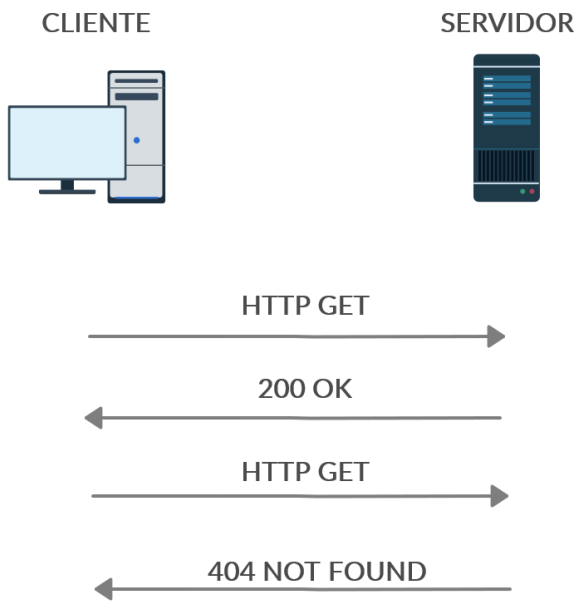

Figura 2-2: Diagrama de flujo de HTTP

El cliente realiza una petición generando un mensaje que contiene el método de la solicitud, la URL del recurso utilizado y la versión del protocolo. Por otro lado, el servidor recibe esta petición y responde al mensaje con un formato similar al de la petición, indicando el estado del objeto pedido.

Por último, HTTP está muy extendido en Internet, y cualquier usuario puede usar un navegador con el que se podrá conectar con cualquier servidor sin tener que realizar

ninguna otra operación más que solicitar una página web como se hace normalmente. Gracias a esto, se ha adoptado HTTP como el protocolo principal para el intercambio de datos entre cliente y servidor.

2.2.3. Protocolo HTTPS

HTTPS (*HyperText Transfer Protocol Secure*) [6] es un protocolo de comunicación de Internet que protege la integridad y la confidencialidad de los datos de los clientes y el servidor. La información que se envía en este protocolo está protegida con el protocolo de seguridad (TLS, *Transport Layer Security*) dando, principalmente, estas capas de seguridad:

- Integridad de los datos: la información no puede modificarse durante la transferencia, ya sea de manera intencionada o no, sin que se detecte.
- Cifrado: toda la información se encripta para asegurar una correcta transmisión de datos.
- Autenticación: el cliente ha de autenticarse mediante un certificado, cuando se verifica la autenticidad del servidor, se envía una clave de sesión que solo éste puede leer. Una vez realizado este paso comenzaría la transmisión de datos de manera segura.

El principal objetivo del protocolo HTTPS es proteger todos los datos que se transmiten a través de la red, ya que a través de HTTP la información puede ser interceptada y es muy susceptible de recibir ataques de terceros. Con el aumento, por ejemplo, de compras por Internet ha ido aumentando el uso de HTTPS para poder proteger la información de tarjetas de crédito.

Un problema que se puede dar en HTTP es el caso de suplantación de identidad, también denominado phishing, con el que se interceptan datos de un cliente y se envían a personas no autorizadas. Con el uso de HTTPS se pueden evitar estos casos, dando así mayor seguridad, privacidad y protección de datos.

Para comprender mejor el uso de este protocolo, en la sección siguiente se estudiará el protocolo de seguridad TLS.

2.2.3.1. Protocolo TLS

El protocolo TLS [7] es una evolución del protocolo obsoleto SSL (*Secure Socket Layer*), ambos son protocolos criptográficos que proporcionan integridad y privacidad en la comunicación entre cliente y servidor, a través del puerto 443, garantizando que dicha información no pueda ser modificada ni interceptada por elementos no autorizados, así solo el cliente y el servidor que se están comunicando serán los que tengan acceso a la comunicación de forma íntegra.

TLS se base en tres fases principales:

- Negociación: cliente y servidor negocian los algoritmos criptográficos que se utilizarán para autenticarse y encriptar los datos. Actualmente existen diversas opciones:
    - Encriptación con clave pública: RSA, DSA, curvas elípticas (ECDHE, son efímeras).
    - Encriptación simétrica: AES (GCM), CHACHA20.
    - Funciones Hash: POLY1305 (MAC), SHA (256, 384, 512).
- Autenticación y claves generadas: cliente y servidor se autentican mediante los certificados y claves intercambiadas en la fase de negociación.
- Transmisión: cliente y servidor pueden comenzar de manera segura con la transmisión de datos cifrada, este tipo de mensajes son denominados *Application Data*.

Actualmente están usándose las versiones 1.2 y 1.3 del protocolo TLS, aunque la mayoría de los navegadores (Google Chrome, Mozilla Firefox y Apple) y servidores web están actualizados para usar la última versión (1.3). A continuación, se detallan y comparan las mejoras de esta última versión respecto a la anterior:

- En la versión 1.3 se ha eliminado el soporte para los siguientes algoritmos debido a sus vulnerabilidades: RC4, RSA, SHA-1, CBC, MD5, IDEA, DES y 3DES. Y se recomienda el uso de las siguientes suites de cifrados:
    - TLS_AES_256_GCM_SHA384
    - TLS_CHACHA20_POLY1305_SHA256
    - TLS_AES_128_GCM_SHA256
    - TLS_AES_128_CCM_8_SHA256
    - TLS_AES_128_CCM_SHA256
- Se ha mejorado el rendimiento del *handshake* inicial aumentando su velocidad debido a que el proceso se completa en un único viaje de ida y vuelta, y no en tres viajes como hacía la versión 1.2. Debido a esta mejor se ha visto reducido el tiempo de latencia. En la siguiente imagen podemos observar esta mejora.

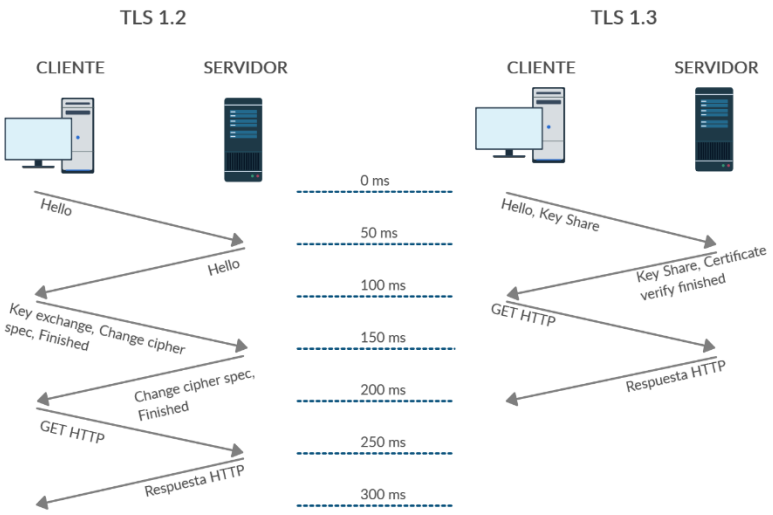

Figura 2-3: Establecimiento de sesión en tres pasos del protocolo TLS

- Otra ventaja es que la versión 1.3 recuerda todo lo visitado. En los sitios web que haya visitado el cliente anteriormente, ahora se puede enviar información desde el primer mensaje al servidor. Esto es lo que se conoce como *Zero Round Trip Time* (0-RTT) y da mejores resultados en los tiempos de carga haciendo que las conexiones sean mucho más rápidas. Pero una desventaja de esto es que abre pequeños agujeros de seguridad, ya que las propiedades de seguridad de esta característica son más débiles que las que se proporcionan de manera regular.
- Por último, se ha producido una mejora en la seguridad debido a que la versión 1.2 en ocasiones no se configuraba de manera adecuada, dejando vulnerabilidades graves en los servidores.

En conclusión, con esta nueva versión del protocolo nos podremos beneficiar durante bastante tiempo de tener conexiones cifradas más seguras y rápidas.

2.3. Funciones de Distribución de Probabilidad

En esta sección se explica la importancia del uso de las distribuciones de Poisson, de Pareto y de las funciones de distribución acumulada (CDF). Este tipo de funciones nos serán de gran utilidad para realizar comparaciones reales sobre las medidas generadas. Primeramente, se generarán las funciones para los tiempos de respuesta obtenidos en cada servidor, posteriormente se aplicarán métodos estadísticos para poder comparar lo registrado por el servidor con la traza de red capturada.

2.3.1. Distribución de Poisson

La distribución de Poisson [13], propuesta por Siméon-Denis Poisson, es una de las más significativas distribuciones de variables discretas y se caracteriza, principalmente, por reflejar la probabilidad de que suceda un determinado número de ocurrencias de un

fenómeno durante un cierto período de tiempo. Esta distribución se emplea para especificar varios procesos como:

- Número de coches que cruzan cierto punto de una carretera durante un período de tiempo definido.
- Número de llamadas realizadas a una central por hora.
- Número de servidores web a los que se accede por minuto.

Su función de probabilidad es definida de la siguiente manera:

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{2.1}$$

Ecuación 2-1: Función de probabilidad de la distribución de Poisson

Donde el parámetro x es el número de ocurrencias de dicho evento y λ el parámetro que describe el número de veces que se espera que se repita el fenómeno durante un intervalo de tiempo dado. A continuación, se muestra un ejemplo de cómo es gráficamente esta distribución.

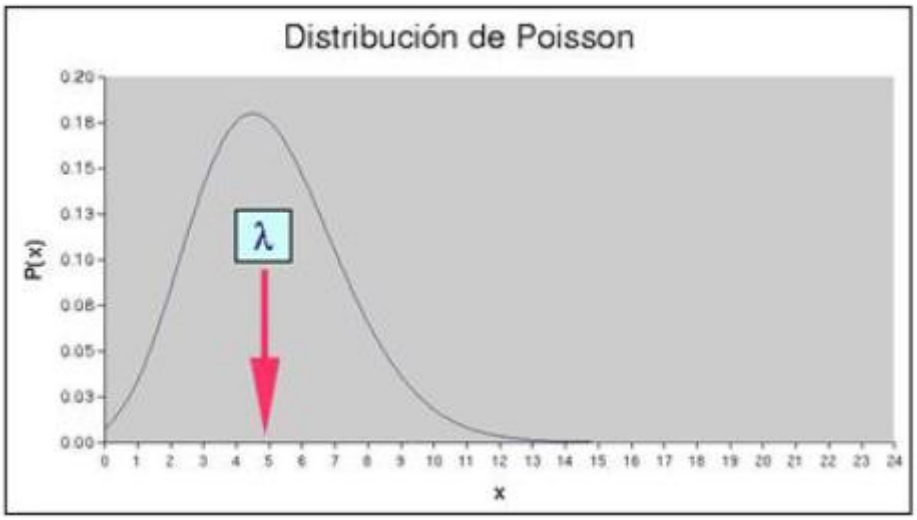

Figura 2-4: Distribución de Poisson con λ igual a 5 [13]

Este tipo de distribución, como se explicará en el Capítulo 5, será útil a la hora de generar tráfico sobre ambos servidores a monitorizar.

2.3.2. Distribución de Pareto

La distribución de Pareto [14], formulada por Vilfredo Pareto, es una distribución de probabilidad biparamétrica continua cuya función de densidad de probabilidad es la siguiente:

$$f(x) = \frac{\alpha\, x_0^{\alpha}}{x^{\alpha+1}} \; donde \; x \geq x_0 \qquad (2.2)$$

Ecuación 2-2: Función de densidad de probabilidad de la distribución de Pareto

Al parámetro x_0 se le conoce como el valor inicial y α es conocido como el índice de pareto.

Gráficamente, la distribución de Pareto se ve de la siguiente manera:

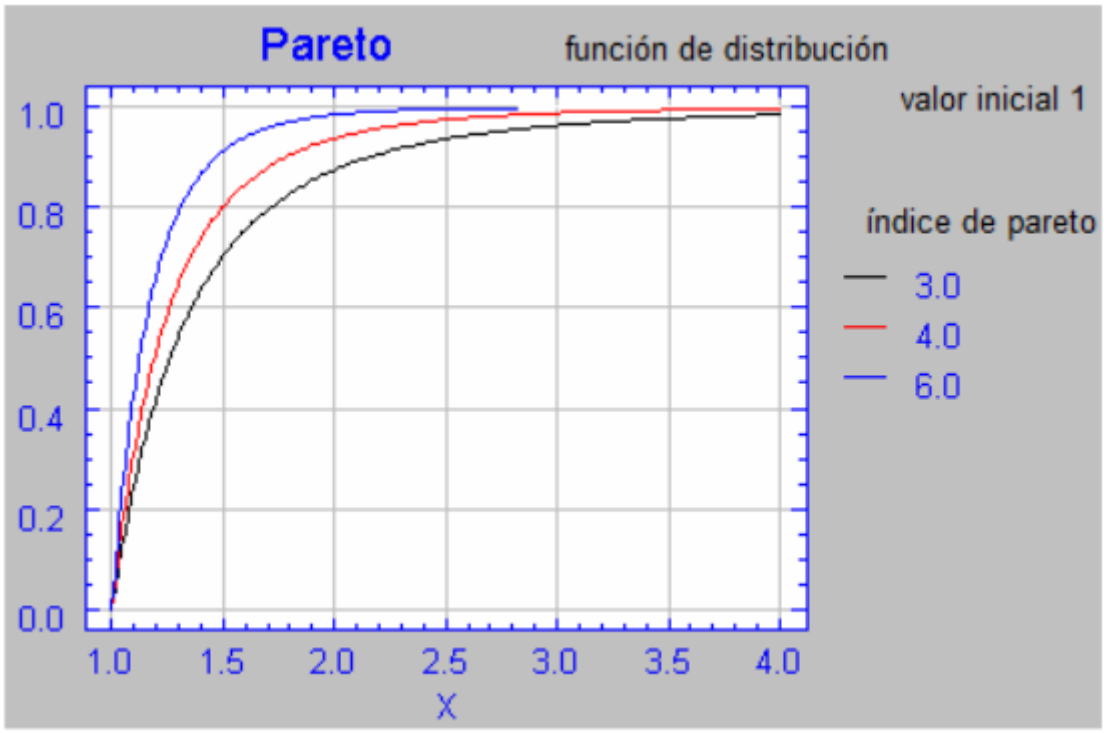

Figura 2-5: Forma gráfica de la distribución de Pareto [14]

Este tipo de distribución surgió por el autor como una manera de calcular la renta de la población, pero también se adecúa a otro tipo de circunstancias y situaciones en las que se establezca una distribución de una magnitud acumulable entre pocos que acumulan grandes cantidades y muchos que acumulan pocas cantidades, por ejemplo, cuando suceden grandes errores en un disco duro frente a pequeños errores.

Esta distribución, como se explicará en el Capítulo 5, será la que sigan los tiempos de llegada de las descargas que se producirán al generar tráfico sobre los servidores.

2.3.3. Función de Distribución Acumulada (CDF)

La función de distribución acumulada (CDF) [15] calcula la probabilidad acumulada de un valor dado de x. Se utiliza este tipo de funciones para determinar la probabilidad de que una observación aleatoria que se toma de los datos sea menor que o igual a cierto valor. También se puede usar esta información para determinar la probabilidad de que una observación sea mayor que cierto valor o se encuentre entre dos valores.

Su definición es la siguiente:

$$F(a) = P(X \leq a) = \sum_{x \leq a} f(x) \tag{2.3}$$

Ecuación 2-3: Definición de la función de distribución acumulada

Como se observa en la ecuación (2.3), *F(x)* es una función que da la proporción de valores de *X* menores o iguales que *x*, para cada valor que contenga *x*. Se encuentra sumando todas las probabilidades hasta el valor *x*.

En este proyecto se utilizarán este tipo de funciones en los que los valores de *x* serán los tiempos de respuesta de las peticiones. Con ello, será fácilmente observable cuáles son los tiempos más altos y qué probabilidades hay de que surjan estos tiempos.

Aparte de realizar una comparación visual sobre las funciones de distribución acumuladas de cada caso, se aplicarán métodos estadísticos para realizar una comparación más exhaustiva que explicaremos en las siguientes secciones.

2.3.4. Estimación de densidad de probabilidad

Para poder utilizar los métodos estadísticos de los Capítulos 2.3.5 y 2.3.6, es necesario estimar la función de densidad de probabilidad para poder realizar dichas comparaciones, ya que las distribuciones generadas son discretas y no continuas, por ello se aplicará este estrategia para poder aplicar dichos métodos.

La estimación que se aplicará será la estimación *kernel* de la densidad (*Kernel Density Estimation*) [16], es un método de estimación no parámetrico que calcula la función de densidad de probabilidad de una cierta variable aleatoria. Las funciones obtenidas de estos métodos cumplen las siguientes características principales:

- Son positivas,
- Son simétricas,
- Y, son continuas

En la Ecuación 2-4 observamos la forma que tiene la función de densidad *kernel*:

$$\hat{f}(h) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{2.4}$$

Ecuación 2-4: Ecuación de la función de densidad *kernel*

Los valores de las muestras de x son aleatorios; K es el *kernel*, y como se ha explicado previamente, es una función no negativa; y h es mayor a cero, ya que es el parámetro de suavizado conocido como ancho de ventana o *bandwidth*. Este *bandwidth* determina el

grado de suavizado que tendrá el *kernel* de modo que a mayor número menor número de picos tendrá la función resultante.

En la Figura 2-6 podemos observar un ejemplo de la función de densidad *kernel*. La función inicial es la de color gris que es una función de distribución normal, y se observa cómo es la función de densidad *kernel* dependiendo del color. La función de color rojo es la resultante con un *bandwidth* muy pequeño y se observa claramente los picos que producen, la negra está generada con un *bandwidth* mayor y la verde es el resultado de aplicar un *bandwidth* bastante grande, en este último caso se aprecia mejor el suavizado de la función.

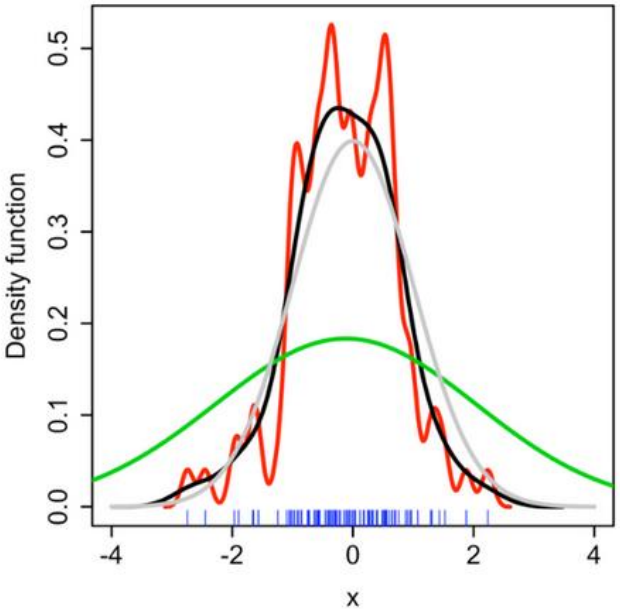

Figura 2-6: Ejemplo de una función de densidad *kernel* [16]

2.3.5. Test Kolmogorov-Smirnov

El método estadístico Kolmogorov-Smirnov [16], también conocido como prueba K-S, se define como una prueba no paramétrica que calcula la distancia vertical máxima entre las muestras de dos funciones de distribución acumulada, la función se muestra a continuación:

$$KS = \max_{x} |F_1(x) - F_2(x)| \tag{2.5}$$

Ecuación 2-5: Ecuación del estadístico de la prueba K-S

La ventaja principal de este método es que es sensible a diferencias tanto en la localización como en la forma de la función de distribución acumulada. Sus principales características son las siguientes:

- Es una prueba de bondad de ajuste, es decir, se utiliza para verificar si las distribuciones obtenidas siguen una distribución normal.
- Esta prueba responde a la pregunta: ¿La distribución de las muestras (distribución empírica) se ajusta a la teórica?
    - En este caso, la hipótesis nula (H0) establecerá que la empírica es similar a la teórica, es decir, que la distribución obtenida de las muestras medidas es consistente con la distribución teórica.
    - Por otra parte, la hipótesis alternativa (H1) establecerá que ambas distribuciones no son consistentes entre sí.

$$\begin{cases} H_0: misma\ distribución \\ H_1: diferente\ distribución \\ \\ H_0: F_1(x) = F_2(x) \\ \\ H_1: F_1(x) \neq F_2(x) \end{cases} \tag{2.6}$$

Ecuación 2-6: Ecuación de la hipótesis de la prueba K-S

- El parámetro alfa indica el nivel de significación de la prueba, siendo 5% el valor por defecto.

Algunas de las ventajas de la prueba K-S es que es más poderosa que la prueba Chi cuadrado, es fácil de calcular y ajustar, y no requiere agrupación de datos.

2.3.6. Divergencia de Kullback-Leibler

La divergencia de Kullback-Leibler [18] es una medida no simétrica de la similitud entre dos funciones de distribución y está muy relacionada con el método de ajuste de distribuciones de máxima verosimilitud. Si tenemos observaciones *x* en una función desconocida *f* y se intenta ajustar a otra función de densidad f_λ, de acuerdo con la teoría de máxima verosimilitud, se busca que el parámetro *y* que maximice la siguiente función:

$$L_\lambda = \sum_i log\, f_\lambda(x_i) \tag{2.7}$$

Ecuación 2-7: Ecuación para calcular la divergencia de Kullback-Leibler

Aproximándola cuando n es grande y realizando la diferencia del término constante, obtenemos la siguiente función:

$$\int f(x)\, log \frac{f(x)}{f_\lambda(x)} \tag{2.8}$$

Ecuación 2-8: Ecuación de la Divergencia de Kullback-Leibler

Esta función resultante es la divergencia de Kullback-Leibler entre f_λ y la función verdadera *f*.

2.3.7. Gráfico cuantil-cuantil (Q-Q Plot)

El gráfico de probabilidad cuantil-cuantil [19] y [20] es un método que permite comparar las distribuciones de un conjunto de datos con una distribución específica.

Los cuantiles son puntos tomados a intervalos regulares de las funciones de distribución de probabilidad, suelen usarse por grupos y dividen la distribución en partes iguales. Los más usados son:

- Percentil: Divide la función en cien partes.
- Decil: Divide la función en diez partes.
- Quintil: Divide la función en cinco partes.
- Cuartil: Divide la función en tres partes.

Este término fue utilizado por vez primera por Kendall, quien definió el cuantil mediante la siguiente ecuación:

$$P(X < x_p) = p;\ siendo\ 0 < p < 1 \tag{2.9}$$

Ecuación 2-9: Ecuación de la definición de cuantil

Conociendo este término podremos aplicar las gráficas cuantil-cuantil para evaluar si un conjunto de datos proviene de una distribución Normal o exponencial. Cuando se van a analizar los datos y asumimos que la distribución a analizar sigue una distribución normal, usando esta gráfica se verificará de manera visual si esta suposición es correcta. Si ambos conjuntos de cuantiles siguen una misma distribución, los puntos se verán formando una línea aproximadamente recta. En la siguiente figura podemos ver un ejemplo claro de una gráfica cuantil-cuantil:

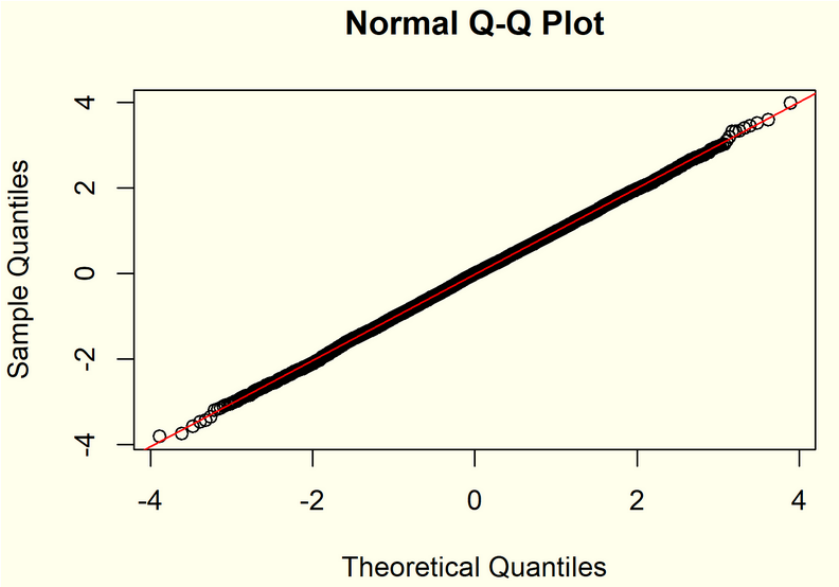

Figura 2-7: Ejemplo de gráfica Q-Q siguiendo una distribución normal [20]

Como se puede observar, en este caso ambas distribuciones seguirían una distribución normal ya que forman una recta prácticamente perfecta. En el caso de que no ocurra esta situación, la gráfica resultante será la que se muestra en la siguiente figura:

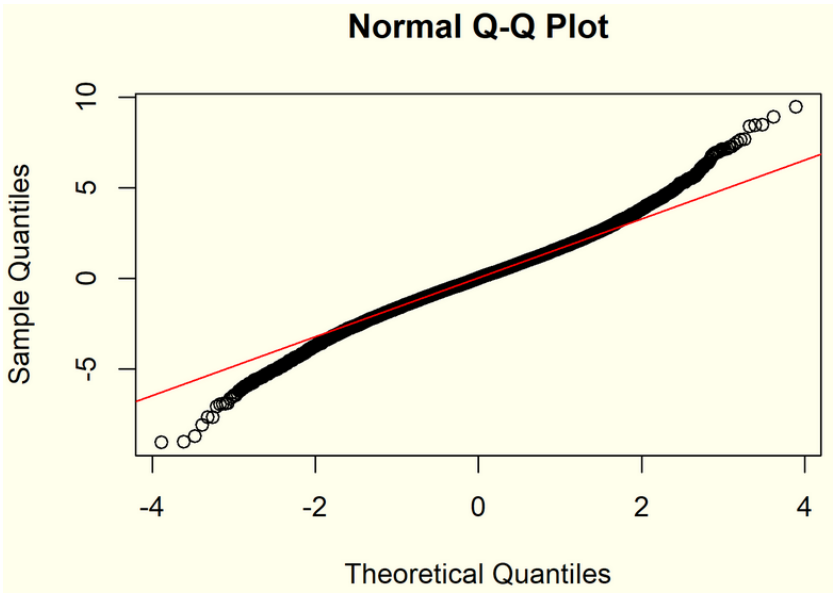

Figura 2-8: Ejemplo de gráfica Q-Q no siguiendo una distribución normal [20]

En este caso al no seguir la misma distribución, los puntos no forman una recta.

2.4. Trabajos Relacionados

En las secciones anteriores se han explicado todos los principios teóricos y métodos estadísticos de este proyecto, por lo que en esta sección trataremos los trabajos relacionados que han motivado la realización de este Trabajo de Fin de Máster. En nuestro caso nos centraremos en trabajos relacionados que se han realizado sobre la

monitorización de los protocolos HTTP y HTTPS. Principalmente, nos centraremos en aquellos que ofrezcan los tiempos de respuesta. Es importante realizar y comprobar dicha medida ya que nos permite identificar qué clientes tienen tiempos altos, con esto, podremos realizar acciones para mejorar el tiempo de respuesta, y a su vez mejorar la calidad de servicio prestada.

En el año 2015, durante la conferencia de software celebrada en Malasia, se mostró un estudio [2] realizado sobre cómo estimar los tiempos de respuesta basándose en los tiempos de llegada que registra el *log* de acceso del servidor HTTP. Consideran que estimar estos tiempos es un factor clave para identificar a los usuarios que están teniendo problemas para ver la página web y poder realizar mejoras en el servidor para que tengan un tiempo de respuesta adecuado. En cuanto a las mediciones que realizan, tienen en cuenta dos puntos de vista: el tiempo que transcurre desde que el cliente realiza la petición y se carga la página completamente, y el tiempo que transcurre desde el inicio de la petición y el inicio de la primera respuesta. A su vez, en el lado del cliente miden el tiempo de respuesta que percibe para poder comparar con lo que registra el servidor. Como conclusión, obtuvieron que los tiempos que registra el servidor no son muy distintos a los tiempos que se capturaron en el lado del cliente, esta diferencia de tiempos se encuentra en el rango de uno a dos segundos e indicaron que esta diferencia es aceptable para tomar medidas correctivas y mejorar la calidad de servicio prestada.

Como se ha mencionado anteriormente, el protocolo HTTPS ha ido tomando mayor importancia en estos últimos años y ha aumentado drásticamente el tráfico de este protocolo. Por ello, el departamento de ingeniería eléctrica, electrónica y de comunicaciones de la Universidad de Navarra [1] propuso una manera de analizar los tiempos de respuesta sobre los servidores HTTPS.

En este artículo tratan sobre la importancia de medir los tiempos de respuesta y la complejidad que añade el protocolo HTTPS para calcular dichos tiempos. Los *logs* de acceso de los servidores son fácilmente analizables ya que contienen los tiempos de respuesta de las peticiones, el problema es que estos *logs* no reflejan el tiempo percibido por el cliente. Primeramente, analizan el tiempo que tarda en realizarse la negociación inicial, el tiempo que tarda este proceso puede afectar al tiempo que percibe el cliente y es también fácilmente medible porque no están encriptados los paquetes. Aunque tienen en cuenta este tiempo, se centran en los tiempos de respuesta de los datos cifrados. Para ello, se tienen en cuenta el número de peticiones y de respuestas que se envían, normalmente las peticiones GET suelen estar contenidas en un solo mensaje *Application Data*, pero, por ejemplo, las peticiones POST requieren varias peticiones cuando se trata de un fichero grande. Por lo tanto, el servidor necesitará enviar varias respuestas dependiendo del contenido pedido.

El método que proponen (*APM tool*) consiste en analizar los *burst*, tanto en HTTP como HTTPS, de los mensajes *Application* Data en cada dirección, asumen que un *burst* es equivalente a una petición o a una respuesta (dependiendo de la dirección). Una petición termina cuando el primer byte de la respuesta está disponible, y la respuesta termina

cuando la conexión se cierra o el primer byte de una nueva petición está disponible. El inicio de la respuesta marca el final de la petición y el inicio de una nueva petición marca el final de la respuesta anterior. Este método lo aplican sobre trazas con *pipeline* que permite el protocolo HTTP/1.1, la Figura 2-9 muestra cómo se envían las peticiones y respuestas. Al estar el tráfico cifrado, el *pipelining* hace que sea más complejo el análisis y puede hacer que surjan problemas en su análisis.

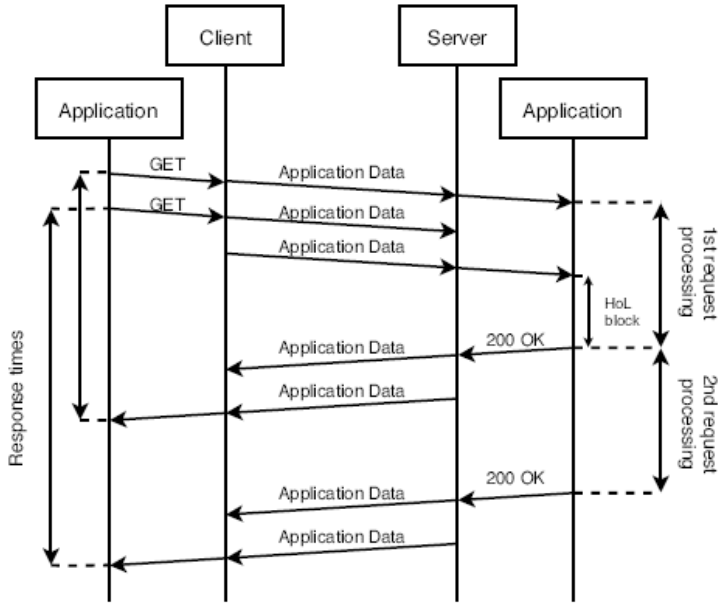

Figura 2-9: Pipelining en HTTP/1.1. [1]

Aun así, obtuvieron resultados mejores de lo que esperaban, obteniendo una gran precisión entre los tiempos medidos con HTTP y HTTPS como se muestra en la siguiente figura.

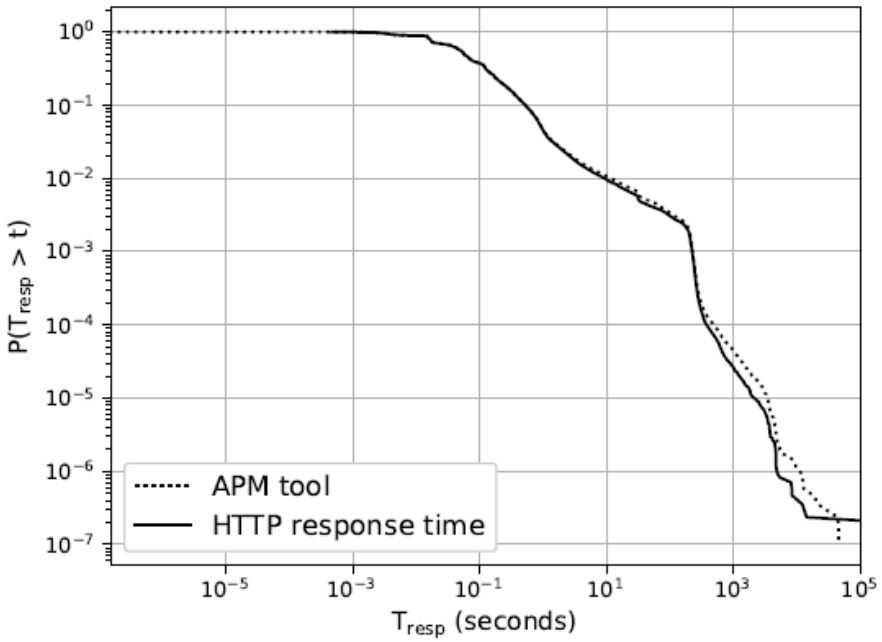

Figura 2-10: Comparación entre los tiempos de respuesta de HTTP y el método propuesto [1]

Sin embargo, el mecanismo de *pipelining* que se ha analizado por parte del departamento de la Universidad de Navarra está siendo abandonado por la mayoría de los navegadores web debido a errores e inconsistencias entre servidores *proxies*.

En septiembre del año 2014, Carlos Vega junto con Paula Roquero y Javier Aracil presentaron un estudio [21] sobre métricas para medir el rendimiento del tráfico de red. Una de estas métricas calculaba los tiempos de respuesta de peticiones HTTP entre el primer paquete de la petición y el primer paquete de la respuesta. Incluso indican que tanto la herramienta Tshark como su herramienta desarrollada calculan de distinta manera los tiempos de respuesta entre peticiones y sus respuestas. Más adelante veremos cómo esta misma situación sucede al comparar los tiempos de respuesta obtenidos en ambos servidores.

En el año 2017, Carlos Vega realizó su tesis [22] sobre el proceso de recolección, análisis y visualización del tráfico de la red. Su objetivo principal fue el de implementar nuevas técnicas en cada una de las etapas que se realizan en la monitorización del tráfico: captura, disección, procesamiento, análisis y visualización de los datos. Para el desarrollo de este Trabajo de Fin de Máster se tomarán como base las técnicas aplicadas para la centralización de los *logs* de acceso de los servidores y cómo se puede realizar una monitorización a través de la metodología ETL (*Extract Transform and Load*) mediante la herramienta Elasticsearch.

Finalmente, una vez analizada la información relacionada con este Trabajo de Fin de Máster sobre la monitorización de los servidores web, nos hemos dado cuenta de que no se ha hecho un análisis sobre los tiempos que contienen los *logs* de acceso de los servidores junto con los tiempos del tráfico HTTPS percibido por parte del cliente.

2.5. Conclusiones

A lo largo de este capítulo se ha explicado el funcionamiento de los principales protocolos de Internet, indicando la importancia que está tomando HTTPS frente HTTP y que será en estos años el protocolo más usado [8].

En cuanto a la monitorización de los servidores se han realizado estudios sobre trabajos previos relacionados con este Trabajo de Fin de Máster. Con ello hemos aprendido sobre cómo monitorizar servidores, tanto HTTP como HTTPS, y qué metodologías se han ido utilizando para comprobar el estado de cada servidor y cómo le afecta al cliente la calidad de servicio prestada.

También se han indicado los métodos estadísticos que se aplicarán en este proyecto y el porqué de la importancia de su utilización.

Finalmente, una vez visto todo el estado del arte de la monitorización de redes, en el siguiente capítulo se documenta cómo se ha realizado el análisis sobre la problemática que ha motivado este proyecto.

3. Análisis del Problema

3.1. Introducción

En este capítulo se analizará el problema que hay en la actualidad con la monitorización del tráfico HTTPS y una posible solución que se ha implementado en este proyecto. El diseño, desarrollo y los resultados de la implementación se mostrará en los capítulos 4 al 6.

3.2. Descripción del Problema

Durante estos años se han realizado numerosos estudios en los que se analizan los tiempos de respuesta para HTTP extrayendo dicha información de los *logs* de acceso de los servidores. Sin embargo, estos *logs* esconden los tiempos de respuesta que perciben los clientes y solamente se pueden medir estos tiempos mediante herramientas de monitorización pasivas. Si a esta problemática se le añade el cifrado que produce el protocolo HTTPS sobre el tráfico, la medida del tiempo de respuesta se vuelve más compleja.

Para nuestro caso, nos centraremos en el tráfico de TCP junto con HTTPS, dejando el análisis del protocolo QUIC para un trabajo futuro. Normalmente las peticiones (HTTP GET) están contenidas un solo mensaje *Application Data*, pero en ocasiones no se da así y esta petición puede requerir enviar varios mensajes, ya sea porque la longitud del contenido es muy grande o por problemas en la red. También hay que tener en cuenta lo mencionado en la sección “2.2.3.1 Protocolo TLS”, dependiendo de qué tenga el servidor implementado se puede tener el caso del 0-RTT y que la petición se envíe en el primer mensaje. Como podemos ver en la Figura 3-1, en cuanto a las respuestas ocurre algo parecido, en general si ha llegado una sola petición el servidor responderá con una única respuesta, aunque en ocasiones esto no suele ocurrir y para responder peticiones se suelen enviar varios mensajes al cliente.

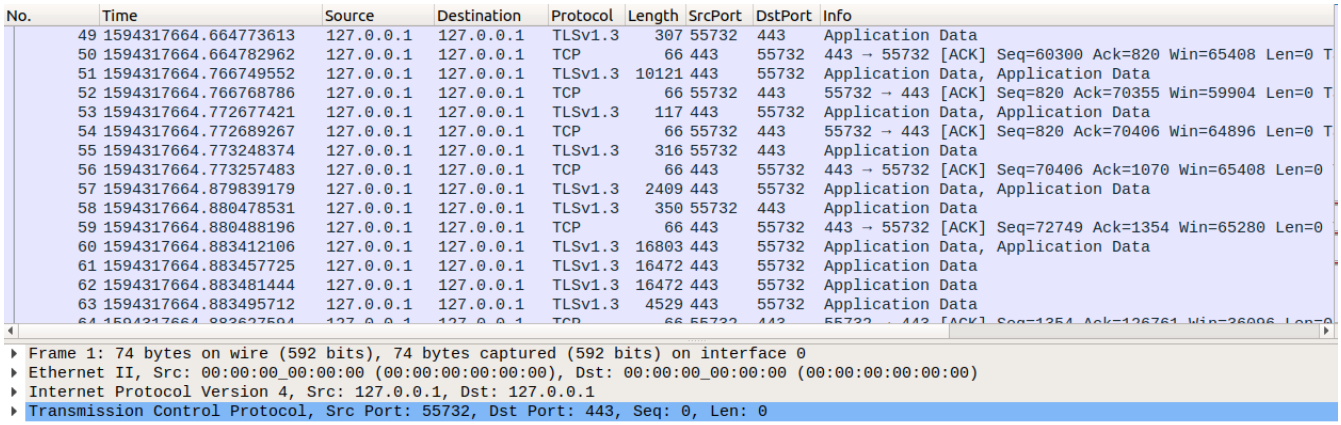

Figura 3-1: Captura de tráfico HTTPS

También hay que tener en cuenta el *handshake* inicial que produce TLS, el cliente lo inicia, pero en él no se incluyen las peticiones, una vez terminada la negociación es cuando se comenzará a generarlas. Con esto queremos reflejar que este tiempo que se tarda en la negociación el cliente lo notará cuando esté realizando peticiones. Este tiempo se puede medir independientemente y se puede tener en cuenta si resulta ser un problema en la conexión. Este proyecto se centrará en los tiempos de respuesta para los datos cifrados, ya que analizar esta negociación es una tarea trivial.

3.3. Propuesta de Solución

La solución propuesta para este Trabajo de Fin de Máster consiste en analizar primeramente los *logs* de acceso al servidor. En la Figura 3-2 podemos ver un ejemplo de lo que contiene este tipo de *logs* en un servidor HTTPS, aunque en la red y mediante herramientas de monitorización visualicemos las peticiones y respuestas encriptadas, el servidor recoge todas las peticiones que recibe indicando, por ejemplo, qué IP las realiza, cuándo llegó dicha petición al servidor o lo que éste tardó en proveer la respuesta.

```
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317662646726 2017706 "GET /wordpress/index.php/contacto/ HTTP/1.1" 200 60299 443 55732 "-" "Wget/1.19.4 (linux-gnu)"
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664664822 108118 "GET /wordpress/index.php/feed/ HTTP/1.1" 200 10106 443 55732 "https://localhost/wordpress/index.php/contacto/" "Wget/1.19.4
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664773296 106777 "GET /wordpress/index.php/comments/feed/ HTTP/1.1" 200 2343 443 55732 "https://localhost/wordpress/index.php/contacto/" "W
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664880548 2995 "GET /wordpress/wp-includes/css/dist/block-library/style.min.css?ver=5.4.2 HTTP/1.1" 200 54012 443 55732 "https://localhost/word
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664884410 613 "GET /wordpress/wp-includes/css/dist/block-library/theme.min.css?ver=5.4.2 HTTP/1.1" 200 2305 443 55732 "https://localhost/wordp
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664885373 852 "GET /wordpress/wp-content/themes/twentyseventeen/style.css?ver=20190507 HTTP/1.1" 200 84438 443 55732 "https://localhost/w
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664886705 706 "GET /wordpress/wp-content/themes/twentyseventeen/assets/css/blocks.css?ver=20190105 HTTP/1.1" 200 10560 443 55732 "https
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664888329 832 "GET /wordpress/wp-content/themes/twentyseventeen/assets/css/colors-dark.css?ver=20190408 HTTP/1.1" 200 18608 443 55732 "
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664890327 1990 "GET /wordpress/wp-includes/js/jquery/jquery.js?ver=1.12.4-wp HTTP/1.1" 200 97351 443 55732 "https://localhost/wordpress/index.
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664897598 3316 "GET /wordpress/wp-includes/js/jquery/jquery-migrate.min.js?ver=1.4.1 HTTP/1.1" 200 10423 443 55732 "https://localhost/wordpres
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317664901001 123239 "GET /wordpress/index.php/wp-json/ HTTP/1.1" 200 85075 443 55732 "https://localhost/wordpress/index.php/contacto/" "Wget/1.
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317665024563 85496 "GET /wordpress/xmlrpc.php?rsd HTTP/1.1" 200 1086 443 55732 "https://localhost/wordpress/index.php/contacto/" "Wget/1.19.4 (
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317665110288 1050 "GET /wordpress/wp-includes/wlwmanifest.xml HTTP/1.1" 200 1381 443 55732 "https://localhost/wordpress/index.php/contacto/" "W
127.0.0.1 127.0.0.1 - - HTTP/1.1 1594317665111610 82788 "GET /wordpress/?p=8 HTTP/1.1" 301 417 443 55732 "https://localhost/wordpress/index.php/contacto/" "Wget/1.19.4 (linux-gnu)"
```

Figura 3-2: *Log* de acceso de un servidor HTTPS

Gracias a la configurabilidad que permiten los servidores de Apache, los *logs* del servidor permiten añadir la opción de lo que el servidor tarda en servir la respuesta, éste no es el tiempo real de respuesta que percibe el cliente, pero será muy próximo a este tiempo. Más adelante se explicará más en detalle este tipo de medida que proporciona Apache.

El siguiente paso será realizar un algoritmo para analizar el *log* junto con las capturas realizadas a través de Wireshark y posteriormente procesadas mediante la herramienta TSTAT. Con esto calcularemos el tiempo de respuesta basándonos en los tiempos que marcan dichas herramientas. Con ello seremos capaces de conocer si los tiempos entre servidor y traza son parecidos o difieren mucho y si los mayores tiempos en cada uno coinciden con la misma IP y puerto, ya que si difieren en esto seremos capaces de relacionar si el problema está en el servidor o en el lado del cliente.

En la Figura 3-3 podemos ver el proceso que se seguirá para desarrollar esta solución:

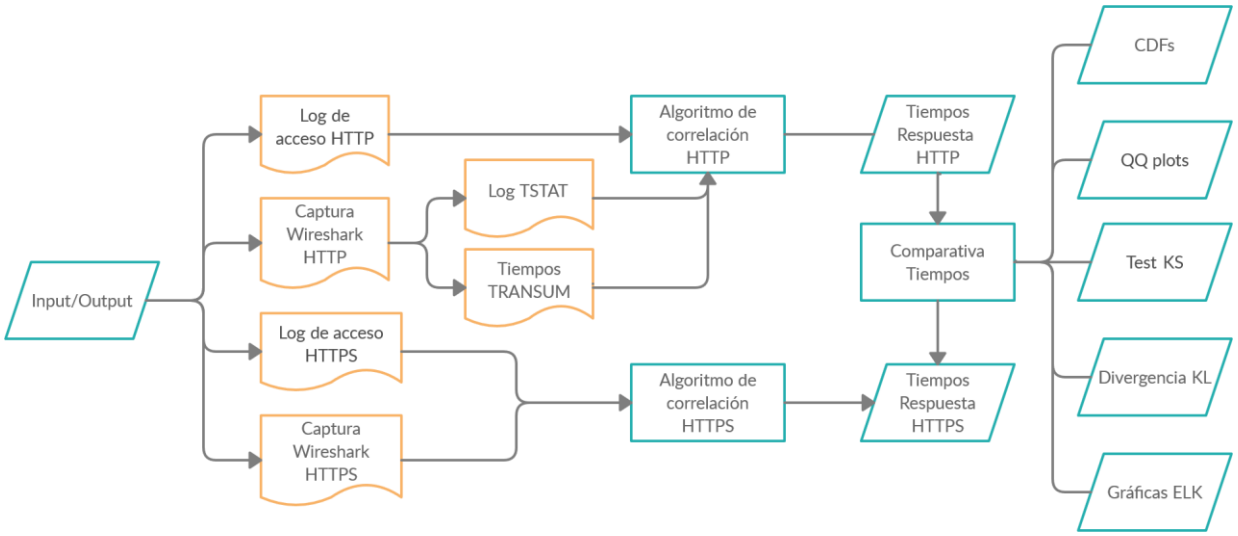

Figura 3-3: Diagrama de Flujo de la Propuesta de Solución

Para validar el correcto funcionamiento de dicho algoritmo de correlación se propondrá el uso de métodos estadísticos para conocer la similitud entre las distribuciones de tiempos de respuesta del servidor y de la traza de red.

Por último, se utilizará el conjunto de herramientas ELK+beats (Elasticsearch, Logstash y Kibana + beats) para analizar los *logs* y las capturas realizadas, este análisis se hará de manera offline, aunque estas herramientas permiten realizar un análisis del servidor en tiempo real. Elasticsearch es un potente motor de búsqueda, Logstash ingesta los *logs* de los servidores sobre Elasticsearch, Kibana es la interfaz gráfica y beats nos ayudará a incluir las capturas realizadas en Elasticsearch. Este conjunto de herramientas está descrito en detalle en el Capítulo 4.3.3.

En los siguientes capítulos se especificará cómo se ha diseñado cada servidor, cómo se ha desarrollado esta propuesta y su validación.

3.4. Conclusiones

Como se ha visto en este capítulo, se ha propuesto un tipo de solución con el principal objetivo de poder monitorizar un servidor de manera efectiva, tanto de manera offline como en tiempo real, dando así la posibilidad de saber dónde puede estar el problema en la conexión y mejorar la calidad de servicio prestada.

4. Diseño del Sistema

4.1. Introducción

En este capítulo se detalla la configuración del sistema en el que se realizará la monitorización de los servidores. Primeramente, se detalla la infraestructura y herramientas de monitorización que se utilizarán, y posteriormente se describe la configuración de cada uno de ellos, tanto de servidores como de las herramientas usadas.

4.2. Infraestructura a Monitorizar

4.2.1. Ubuntu

Ubuntu [23] es una distribución gratuita y de código abierto basada en el sistema operativo Debian GNU/Linux. Su filosofía es que todo el mundo pueda utilizarlo y cuenta con la colaboración de diversos desarrolladores que se encargan de mantener actualizado este sistema operativo, añadiéndole nuevas características y optimizaciones.

Ubuntu destaca frente a otras distribuciones Linux debido a su sencillez de uso y que provee de un gran entorno de programación que hace que sea más fácil trabajar con cualquier tipo de lenguaje. Al principio fue usado solamente para entornos domésticos, pero según muestran las estadísticas de w3techs [24], Ubuntu es la distribución más usada para los servidores web.

4.2.2. Oracle VM Virtualbox

Oracle VM Virtualbox [25] es un software de virtualización multiplataforma desarrollado por Oracle el cual permite instalar sistemas operativos adicionales (invitados) dentro de un sistema operativo anfitrión, cada uno teniendo su entorno independiente.

El sistema operativo anfitrión es donde será instalado el programa Virtualbox y permitirá ejecutar otros sistemas operativos adicionales dentro de él y de manera simultánea, por ejemplo, ejecutar el sistema operativo Ubuntu (invitado) sobre Windows (anfitrión).

Virtualbox también ofrece un paquete especial llamado VirtualBox Guest Additions, este paquete es un software que ha de instalarse en cada máquina virtual creada para optimizar su rendimiento y añadir nuevas funcionalidades. Las Guest Additions ofrecen diversas funcionalidades, pero destacaremos las siguientes:

- Sincronización horaria: Gracias a este paquete, el programa VirtualBox se asegura que la hora del sistema virtualizada y esté mejor sincronizada. Con esto nos aseguramos de que no exista tanto desfase entre el reloj del anfitrión y el reloj del servidor.
- Carpetas compartidas: Con esto se pueden intercambiar datos rápidamente entre el sistema real y el sistema virtualizado.

Con esta herramienta de virtualización se crearán los servidores web sobre el sistema operativo Ubuntu 18.04 LTS.

4.2.3. Apache

Apache es una organización descentralizada y sin ánimos de lucro que da soporte a los proyectos de software bajo denominación Apache, en el que se incluye el popular servidor HTTP Apache [26]. Sus proyectos son de código abierto y se caracterizan por un modelo de desarrollo basado en la colaboración y en una licencia de software abierta.

Para este proyecto se ha utilizado el anteriormente mencionado servidor HTTP de Apache, este es un servidor web gratuito y el 46% [27] de los sitios web del mundo están basados en este tipo de servidor. Aunque comúnmente se le denomina servidor web, no es un servidor físico, sino un software que se ejecuta en un servidor. Su principal tarea es la de establecer una conexión entre un servidor y los navegadores de los visitantes del sitio web mientras envían archivos entre ellos siguiendo la estructura cliente-servidor.

Cuando un cliente quiere ver el contenido de tu sitio web, su navegador le envía una petición a tu servidor y le devuelve una respuesta con todos los archivos solicitados. El servidor y el cliente se comunican a través de los protocolos HTTP o HTTPS y Apache es responsable de garantizar una comunicación fluida y segura entre las dos máquinas. Todo este tráfico HTTP queda registrado en el *log* de acceso que provee Apache y si ocurre algún error también queda registrado en el *log* de errores.

Este servidor creado es altamente personalizable, ya que tiene una estructura basada en módulos que permiten a los administradores configurar las funcionalidades que requieran y desactivar las que no se usen.

Además de apache existen otros servidores web, por lo que a continuación, se muestra una gráfica comparativa entre ellos indicando las ventajas e inconvenientes de cada uno:

Servidores Web	Ventajas	Inconvenientes
Apache	▪ Código abierto y gratuito, incluso para uso comercial ▪ Software estable y confiable ▪ Actualizaciones regulares ▪ Fácil de configurar para principiantes ▪ Multiplataforma ▪ Listo para trabajar con WordPress ▪ Soporte disponible	▪ Problemas de rendimiento con demasiado tráfico ▪ Demasiadas opciones de configuración pueden generar vulnerabilidades
NGINX	▪ Creado para resolver el problema c10k (10.000 conexiones concurrentes) ▪ Mayor escalabilidad debido al modelo basado en eventos ▪ Utilizado en sitios web con alto tráfico como Netflix o Pinterest ▪ Multiplataforma	▪ Incapacidad de utilizar ficheros de configuración ▪ Falta de disponibilidad de mod_php
Tomcat	▪ Creado para aplicaciones Java ▪ No requiere muchos recursos	▪ Poco configurable ▪ Poco eficiente para páginas web estáticas y grandes aplicaciones
Microsoft IIS	▪ Procesamiento de páginas en ASP, Perl o PHP ▪ Ofrece servicios SFTP y FTP ▪ Se integra con Microsoft Azure	▪ Exclusivo de Windows

Figura 4-1: Comparativa de Servidores Web

Analizando estas ventajas e inconvenientes de estos servidores web se ha escogido instalar del de Apache debido a que es de código abierto, fácilmente configurable y multiplataforma.

4.3. Herramientas de Monitorización

4.3.1. Wireshark

Wireshark [28] es uno de los más populares analizadores multiplataforma de protocolos de red que existen. Es una herramienta gráfica usada para analizar e identificar el tráfico de una interfaz de red en un momento concreto. Sus principales características son las siguientes:

- Permite obtener información detallada de los protocolos utilizados en la captura de red.
- Dispone de una amplia gama de filtros que facilita la búsqueda de protocolos.
- Interfaz gráfica sencilla e intuitiva.
- Cuenta con la posibilidad de importar/exportar los paquetes capturados desde/hacia otros programas.

También incluye una lista bastante extensa de protocolos adicionales que es capaz de analizar, en este proyecto se utilizará el plugin adicional del protocolo TRANSUM el cual calcula el tiempo que transcurre entre el inicio de la petición y el final de la respuesta.

Existen otros tipos de analizadores de protocolos de red como EtherApe, tcpdump, Kismet, pero no nos ofrecen la posibilidad de analizar los paquetes con tanto nivel de detalle o suelen carecer de la flexibilidad que ofrece Wireshark.

4.3.2. TCP STatistic and Analysis Tool (TSTAT)

La herramienta TSTAT [29] es un analizador de paquetes pasivo capaz de proporcionar información sobre los patrones de tráfico tanto en la red como en los niveles de transporte.

La falta de herramientas automáticas capaces de analizar datos estadísticos a partir de trazas de paquetes de red fue una motivación importante para desarrollar la herramienta TSTAT, con la que a partir de bibliotecas estándar de software ofrece a los administradores de red e investigadores examinar la información importante sobre índices de rendimiento clásicos y actuales y datos estadísticos. sobre el tráfico de internet. Fue iniciado como una evolución de la herramienta tcptrace, TSTAT analiza los paquetes capturados en tiempo real. Además de la captura en vivo, TSTAT puede analizar los paquetes capturados admitiendo diversos tipos de formatos como libpcap.

En un principio se barajó utilizar otras herramientas de análisis de trazas de red como YAF (Yet Another Flowmeter), pero se observó que la herramienta TSTAT era más potente y permitía obtener más información de los paquetes de red.

4.3.3. ELK Stack

ELK [30] es la sigla para tres proyectos que son open source: Elasticsearch, Logstash y Kibana. Elasticsearch es un motor de búsqueda desarrollado en Java, Logstash es un pipeline de procesamiento de datos del lado del servidor que ingesta datos de una multitud de fuentes simultáneamente, los transforma y luego los envía a Elasticsearch, por último, Kibana es la interfaz gráfica que permite a los usuarios visualizar los datos con Elasticsearch.

A continuación, se mostrarán las principales características de cada proyecto:

- Elasticsearch
    - Su parte core está basado en Lucene.
        - Lucene es una librería de búsqueda de texto de alto rendimiento y basado en índices invertidos.
    - Es orientado a documentos: Utiliza JSON.
    - Realiza escalado de manera dinámica.
    - Permite búsquedas estructuradas como no estructuradas.
- Logstash
    - Las entradas son las fuentes de datos que se procesarán posteriormente en elasticsearch, está centrado en la carga de *logs* de los servidores.
    - Permite realizar filtros para procesar los datos que interesen al administrador.
    - Permite cargar datos en varios servidores.
- Kibana
    - Es una interfaz de usuario gratuita y abierta que permite visualizar los datos ingestados en Elasticsearch.

- Realiza lo que uno quiera, desde rastrear la carga de búsqueda hasta entender la forma en que las solicitudes fluyen por las aplicaciones.
- Permite crear todo tipo de gráficos, desde histogramas a mapas mostrando la geolocalización del tráfico.

En ocasiones también se utiliza el proyecto *filebeat*, perteneciente al conjunto de proyectos que forman *beats* y que desarrolla la empresa *elastic*, junto a ELK para enviar *logs o capturas* a Logstash para procesarlos o directamente a Elasticsearch.

En la siguiente figura mostraremos la arquitectura ELK+beats que sigue este Trabajo de Fin de Máster:

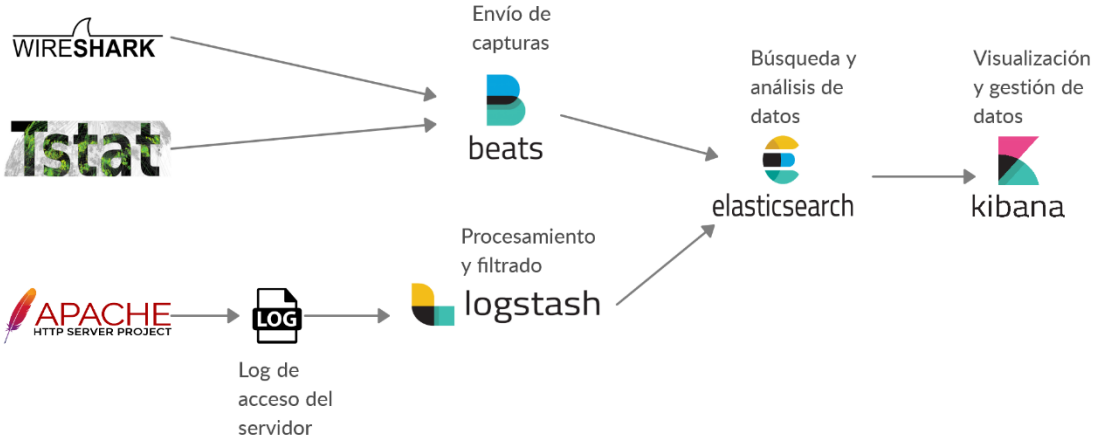

Figura 4-2: Arquitectura ELK + Beats desplegada en el proyecto

Partimos de los *logs* de acceso que generan los servidores web Apache HTTP y HTTPS, mediante Logstash se hace una ingesta de estos *logs* a Elasticsearch, a su vez, las capturas realizadas con las herramientas Wireshark y TSTAT se envían a través de Beats a Elasticsearch, éste se encarga de almacenarlos y crear los índices conteniendo la información de cada *log* y captura, y, por último, Kibana se encarga de analizar y mostrar los datos.

4.4. Configuración de la Infraestructura

A lo largo de esta sección se definirá la configuración que se ha hecho sobre cada servidor y cómo se han configurado las herramientas para obtener los datos necesarios para su posterior análisis.

4.4.1. Configuración de los Servidores Web

4.4.1.1.Servidor Web HTTP

El primer paso llevado a cabo ha sido el de crear y configurar el servidor HTTP. Para ello, se han llevado a cabo los siguientes pasos:

1. Descarga de Ubuntu y del software de virtualización que ofrece Oracle (Virtualbox).
2. Instalación de Ubuntu sobre Virtualbox junto con el paquete GuestAdditions.
3. Instalación del servidor web HTTP de Apache

```
tfmhttp@tfmhttp-VirtualBox:~$ sudo apt install apache2
```

Figura 4-3: Comando para instalar el servidor web HTTP de Apache

4. Configuración del *log* de acceso del servidor.
   a. Modificamos el fichero de configuración de apache, que se encuentra en la ruta /etc/apache2/apache2.conf, mediante la directiva mod_log_config para que el servidor tenga en cuenta los datos que necesitaremos.

```
LogFormat "%h %a %l %u %H %{usec}t %D \"%r\" %>s %O %p %{remote}p \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

Figura 4-4:Configuración del formato del *log* de acceso del servidor

A continuación, explicaremos lo que significa cada atributo:
%h: Dirección IP del servidor.
%a: Dirección IP del cliente.
%l: Nombre de registro remoto. Normalmente aparecerá un guion a menos que se configure el módulo mod_ident de Apache.
%u: Nombre del usuario en el caso de que sea necesaria su autenticación.
%{usec}t: Tiempo, en microsegundos, en el que llega el comienzo de la petición.
%D: Tiempo, en microsegundos, que tarda el servidor en servir la petición.
\"%r"\: Primera línea de la petición.
%>s: Estado final de la respuesta.
%O: Bytes enviados, incluyendo la cabecera.
%p: Puerto del servidor.
%{remote}p: Puerto del cliente.
\"%{Referer}i\": Dirección de la página web pedida.
\"{User-Agent}i\": Contiene el agente de usuario, suelen indicarse los navegadores web que utilizan los clientes.

5. Instalación, configuración y diseño de una página web a través de wordpress.

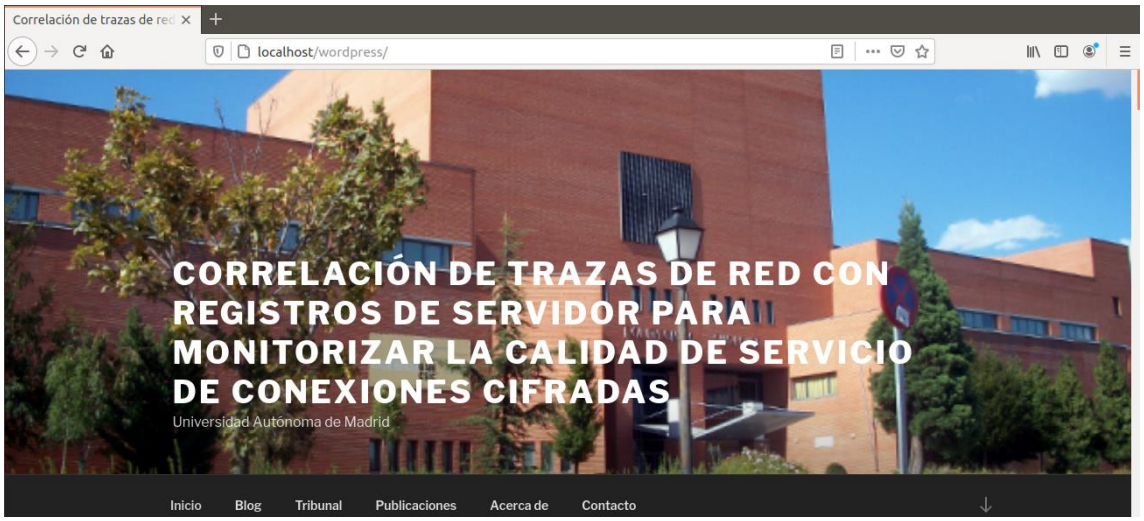

Figura 4-5: Página web diseñada para el proyecto

a. En esta página web se han incluido imágenes, vídeos y ficheros para que el servidor sea lo más parecido a uno real.

6. Por último, se instalará el Wireshark y TSTAT.

Con todos estos pasos tenemos listo el servidor y está preparado para empezar la monitorización sobre él.

4.4.1.2. Servidor Web HTTPS

La configuración de este servidor se ha hecho igual que todos los pasos que se hicieron sobre el servidor HTTP, pero para que este servidor esté cifrado se han realizado los siguientes pasos:

1. Creación de la clave y del certificado:
   a. Instalamos openssl. Es una librería de software para los protocolos TLS y SSL y necesaria para configurar el servidor HTTPS.
   b. Accedemos a la ruta /etc/apache2 y ejecutamos el siguiente comando para generar la clave RSA con una extensión de 2048 bits (necesitamos permisos de superusuario para generar):

      sudo openssl genrsa -out LLaveSSL.key 2048

   c. Ahora podemos generar el certificado a través del siguiente comando:

      sudo openssl req -new -key LLaveSSL.key -out miCertificado.csr

   d. Después de introducir todos los datos que se van pidiendo, toca firmar el certificado. Para ello ejecutaremos el siguiente comando:

      sudo openssl x509 -req -days 365 -in miCertificado.csr -signkey LLaveSSL.key -out miCertificadoFirmado.crt

La opción x509 indica que deseamos crear un certificado autofirmado en vez de generar una solicitud de firma de certificados.
Y la opción -days 365 indica el tiempo por el cual durará dicho certificado.

2. Habilitación de SSL en el servidor.
   a. Una vez generado el certificado habilitamos que nuestro servidor pueda trabajar con SSL a través del siguiente comando:

      sudo a2enmod ssl

   b. Copiamos la clave generada al siguiente directorio que se ha creado para que el servidor trabaje con SSL.

      sudo cp LLaveSSL.key /etc/ssl/private

   c. Y también copiamos el certificado generado al siguiente directorio:

      sudo cp miCertificadoFirmado.crt /etc/ssl/certs

   d. Ahora accederemos a la siguiente ruta y procederemos a modificar el fichero por defecto que genera SSL.

      cd /etc/apache2/sites-available
      sudo gedit default-ssl.conf

   e. Modificamos el fichero default-ssl.conf con la siguiente información:

      Se han incluido las tres líneas posteriores a SSLEngine on y se han comentado las dos últimas.

```
#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on
SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
SSLCertificateFile /etc/ssl/certs/miCertificadoFirmado.crt
SSLCertificateKeyFile /etc/ssl/private/LLaveSSL.key

#   A self-signed (snakeoil) certificate can be created by installing
#   the ssl-cert package. See
#   /usr/share/doc/apache2/README.Debian.gz for more info.
#   If both key and certificate are stored in the same file, only the
#   SSLCertificateFile directive is needed.
#SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
#SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

Figura 4-6: Contenido del fichero default-ssl.con

3. El siguiente paso es instalar un plugin en wordpress para asegurar que funcione correctamente con el protocolo HTTPS. Para ello se instala el plugin Really Simple SSl.
4. Por último, activamos esta nueva configuración y reiniciamos el servidor web.

   sudo a2ensite default-ssl.conf
   sudo service apache2 restart

5. Con esto podemos acceder ya a nuestro servidor web cifrado como muestra la siguiente figura:

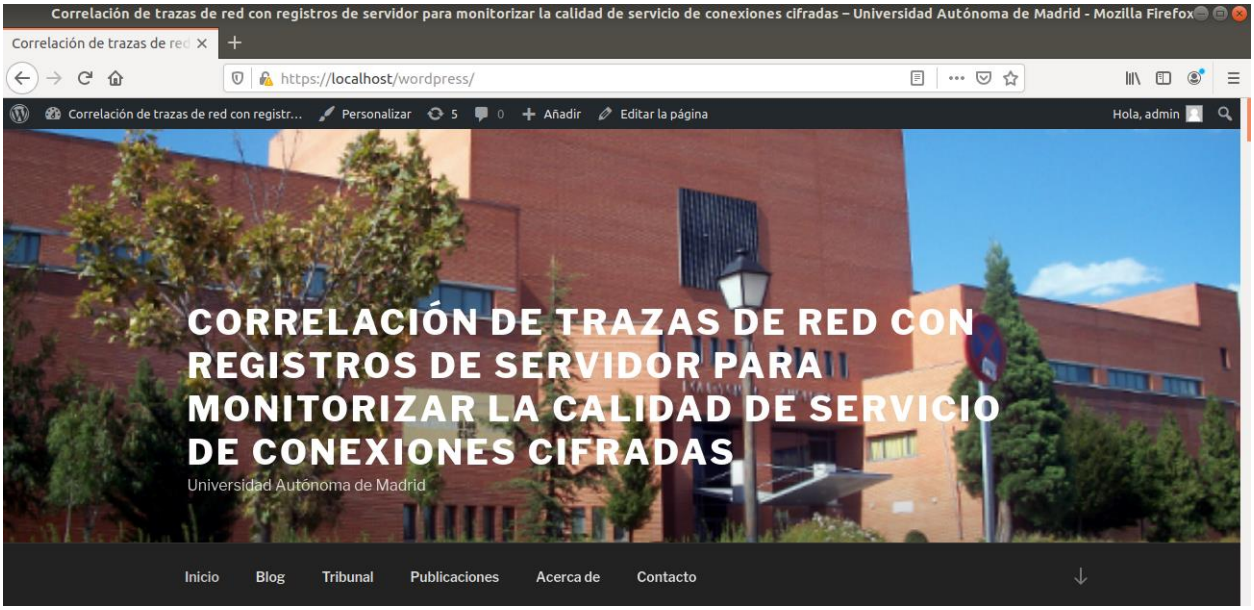

Figura 4-7: Página web principal del servidor HTTPS

4.4.2. Configuración de las Herramientas de Monitorización

Una vez configurados los servidores procederemos a configurar las herramientas que se utilizarán para monitorizar ambos servidores.

4.4.2.1. Configuración de Wireshark

La configuración de la herramienta Wireshark consiste en habilitar todas las interfaces para poder capturar sobre la interfaz Loopback:lo (localhost) y añadimos dos columnas que nos indicarán los puertos de envío y de recepción.

El siguiente paso es habilitar el plugin TRANSUM con la opción de que se calcule desde la posición de cliente, ya que se realizará la captura desde aquí y nos interesa el tiempo de respuesta que percibe el cliente. Este plugin calcula el tiempo que transcurre entre el inicio de la petición y el final de la respuesta. Esto solamente se podrá activar para el servidor HTTP ya que en el servidor HTTPS todas las peticiones/respuestas están cifradas y no es posible realizar esta medida.

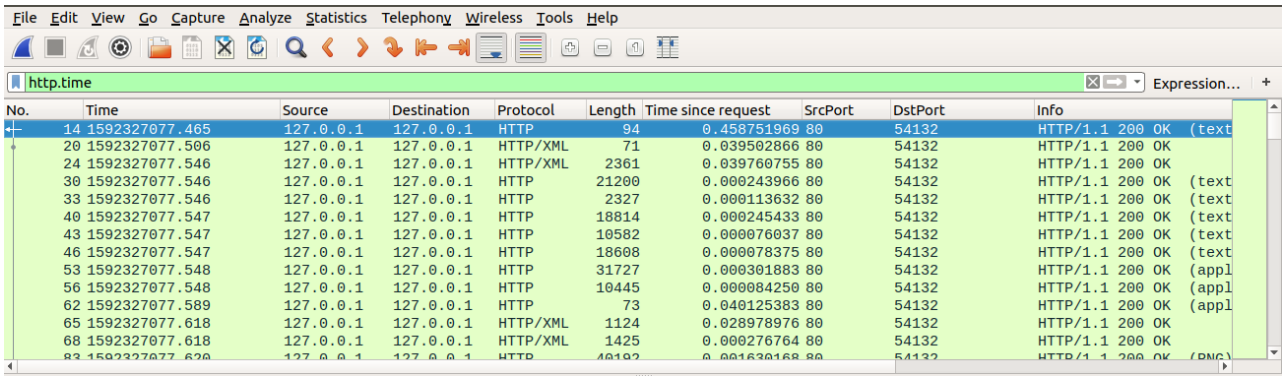

Figura 4-8: Plugin TRANSUM que proporciona Wireshark

Como podemos ver en la gráfica, gracias a esta opción que contiene Wireshark podemos obtener esta medida fácilmente y de una manera simple, añadiendo un filtro sencillo obtenemos estos tiempos.

4.4.2.2. Configuración de TSTAT

En cuanto a la configuración de la herramienta TSTAT tenemos que editar su fichero de configuración *runtime.conf*. En él se puede activar o desactivar los distintos *logs* que nos ofrece. Para nuestro caso, como vemos en la Figura 4-9, activaremos los *logs* de TCP, HTTP, HTTPS (tcplog_layer7) y se indica también que muestre de manera parcial la URL pedida por el cliente, esta opción es más que suficiente para saber a qué están accediendo los clientes en el servidor.

```
log_tcp_complete = 1            # tcp connections correctly terminated
log_tcp_nocomplete = 1          # tcp connections not properly terminated
log_udp_complete = 0            # udp flows
log_mm_complete = 0             # multimedia
log_skype_complete = 0          # skype traffic
log_chat_complete = 0           # MSN/Yahoo/Jabber chat flows
log_chat_messages = 0           # MSN/Yahoo/Jabber chat messages
log_video_complete = 0          # video (YouTube and others)
log_http_complete = 1           # all the HTTP requests/responses

# log options
[options]
tcplog_end_to_end = 0           # Enable the logging of the End_to_End set of measures (RTT, TTL)
tcplog_layer7 = 1               # Enable the logging of the Layer7 set of measures (SSL cert., message counts)
tcplog_p2p = 0                  # Enable the logging of the P2P set of measures (P2P subtype and ED2K data)
tcplog_options = 0              # Enable the logging of the TCP Options set of measures
tcplog_advanced = 0             # Enable the logging of the Advanced set of measures

videolog_end_to_end = 0         # Enable the logging in log_video_complete of the TCP End_to_End set of measures (RTT, TTL)
videolog_layer7 = 0                     # Enable the logging in log_video_complete of the Layer7 set of measures (SSL cert., message counts)
videolog_videoinfo = 0          # Enable the logging in log_video_complete of the additional video info (resolution, bitrate)
videolog_youtube = 0            # Enable the logging in log_video_complete of the YouTube specific information
videolog_options = 0            # Enable the logging in log_video_complete of the TCP Options set of measures
videolog_advanced = 0           # Enable the logging in log_video_complete of video-related Advanced mesurements (rate)

httplog_full_url = 1            # Enable the logging of the partial (=1) or full (=2) URLs in log_http_complete
```

Figura 4-9: Archivo de configuración de la herramienta TSTAT

La herramienta Tstat se aplicará de manera offline, es decir, se ejecutará sobre las capturas generadas con Wireshark para realizar su posterior análisis y correlarlo con los *logs* de acceso del servidor, ya que la captura en vivo del Tstat no es tan eficiente como la realizada por Wireshark.

4.4.2.3. Configuración de ELK + Filebeat

En cuanto a este conjunto de proyectos, solamente ha sido necesario configurar Logstash para poder procesar correctamente el *log* de acceso del servidor. Logstash tiene una configuración predeterminada para los *logs* de Apache, pero al haber modificado el *log* para mostrar más información y con los tiempos en microsegundos se ha tenido que crear un fichero de configuración, este fichero se ha llamado apache2.conf y su contenido es el siguiente:

```
input {
    file {
        path => "/home/tfmhttp/Documentos/Capturas/23072020_tbf_1mbit_50ms/access23072020.log"
        start_position => "beginning"
        sincedb_path => "/dev/null"
    }
}
filter {
    grok {
        match => {"message" => "%{IP:host} %{IP:client} %{USER} %{USERNAME} %{NUMBER:timestamp} %{NUMBER:reqtime} (?:%{WORD:verb}
        %{NOTSPACE:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest}) %{NUMBER:response} %{NUMBER:bytes} %{POSINT:port} %{POSINT:client_port} %{QS:referrer} %{QS:agent}"}
    }
    mutate {
        convert => {
            "timestamp" => "integer"
            "reqtime" => "integer"
            "response" => "integer"
            "bytes" => "integer"
            "port" => "integer"
            "client_port" => "integer"
        }
    }
}
output {
    elasticsearch {
        hosts => ["http://localhost:9200"]
        index => "apachelog"
    }
}
```

Figura 4-10: Fichero de configuración para Logstash

Primeramente, se le indica qué fichero *log* se quiere procesar, después, para que sepa etiquetar y clasificar los campos hay que añadir un filtro a través del plugin grok. Este filtro clasifica todo como un string y Kibana no es capaz de realizar visualizaciones con este tipo de datos, por lo que a través de la función mutate indicamos qué campos queremos que no sean string y los convertimos a un entero. Por último, indicamos que la salida del Logstash vaya a Elasticsearch y creamos el índice apachelog para poder visualizar y filtrar a través de Kibana.

Las capturas que se realizan mediante Wireshark y los *logs* procesados por TSTAT los convertiremos a csv y se procesarán a través de Filebeat. Esta herramienta permite procesar de manera automática los ficheros csv sin necesidad de indicar los campos que contienen.

4.5. Conclusiones

En este capítulo hemos visto cuáles serán las infraestructuras a monitorizar y qué herramientas se usarán para ello. Estas herramientas han sido escogidas de acuerdo con las ventajas que proporcionan frente a otras y que hacen que este proyecto se pueda desarrollar de manera más sencilla.

5. Desarrollo de la Solución

5.1. Introducción

En esta fase se define el proceso de desarrollo que se ha realizado para las infraestructuras a monitorizar. Primero, explicaremos cómo se han generado las bases de datos de tráfico de red y qué tipo de información contienen. Por último, se explicará cómo se han implementado los algoritmos de correlación para ambos servidores, siendo esta etapa la más importante de este Trabajo de Fin de Máster.

5.2. Bases de Datos Generadas

Una vez montados y configurados ambos servidores, lo primero que se ha realizado ha sido generar tráfico web sobre ellos. Como se han creado desde cero y en este proyecto no se van a monitorizar *logs* de servidores reales junto con sus trazas de red, se ha procedido a diseñar un script que simule peticiones a los servidores. Como se indica en el artículo [31] realizado por los departamentos de telemática de las universidades de Granada y Sevilla, la generación de conjuntos de datos de trazas de red que sean lo más parecidos a un tráfico real es una tarea compleja. Estos conjuntos de datos creados deben tener las siguientes características:

- Han de ser realistas o lo más realista posible.
- Tienen que incluir una gran cantidad de volumen de datos para que sea posible realizar entrenamientos o validaciones sobre dicho conjunto.
- Tiene que ser fácilmente actualizable, esto es, que a la hora de generar el tráfico y sea necesario modificar algún parámetro, sea fácilmente configurable.
- Y, por último, los conjuntos de datos han de estar etiquetados, especificando para qué se va a utilizar dicho conjunto.

Teniendo en cuenta estas características, para la generación del script se han estudiado diferentes métodos para enviar peticiones al servidor, siendo el método escogido el de aplicar un *web crawler* sobre los servidores. Un *web crawler*, también conocido como araña o rastreador web, es una herramienta que recorre por completo un sitio web hasta que todas las páginas se han indexado.

Estos rastreadores recogen información como, por ejemplo, la URL del sitio web, el contenido de dicha página, los enlaces y los destinos de dichos enlaces, y cualquier otro tipo de información relevante.

En la Tabla 5-1 podemos observar la comparativa realizada sobre las características más importantes de los *web crawlers* estudiados que nos ha servido para saber cuál utilizar.

Tabla 5-1: Comparativa entre web crawlers

Características	wget	curl	pavuk
Descarga recursiva	SÍ	NO	SÍ
Múltiples URLs	SÍ	SÍ	SÍ
Reintentar descargas fallidas	SÍ	SÍ	SÍ
Certificados SSL cliente	SÍ	SÍ	NO
HTTP POST	SÍ	SÍ	SÍ
HTTP / HTTPS / FTP	SÍ	SÍ	SÍ
FTPS / SFTP / LDAP / POP3 / IMAP	NO	SÍ	NO
Decisión Final	SÍ	NO	NO

Al principio se escogió la herramienta *curl* ya que soporta más protocolos y dispone de más funcionalidades, pero para generar peticiones y debido a la descarga recursiva que genera el comando *wget* se optó finalmente por esta opción. En la siguiente figura podemos ver un ejemplo de lo que se registra en el *log* de acceso del servidor usando el comando *wget* para realizar una sola petición GET sobre una página del servidor creado.

```
127.0.0.1 127.0.0.1 - - 1592327077007313 459184 "GET /wordpress/index.php/blog/ HTTP/1.1" 200 66372 80 54132 "-" "Wget/1.19.4 (linux-gnu)"
127.0.0.1 127.0.0.1 - - 1592327077466623 39596 "GET /wordpress/index.php/feed/ HTTP/1.1" 200 10006 80 54132 "http://localhost/wordpress/index.php/blog/" "Wget/1.
127.0.0.1 127.0.0.1 - - 1592327077506331 39917 "GET /wordpress/index.php/comments/feed/ HTTP/1.1" 200 2295 80 54132 "http://localhost/wordpress/index.php/blog/"
127.0.0.1 127.0.0.1 - - 1592327077546382 391 "GET /wordpress/wp-includes/css/dist/block-library/style.min.css?ver=5.4.2 HTTP/1.1" 200 53902 80 54132 "http://loca
127.0.0.1 127.0.0.1 - - 1592327077546790 148 "GET /wordpress/wp-includes/css/dist/block-library/theme.min.css?ver=5.4.2 HTTP/1.1" 200 2261 80 54132 "http://local
127.0.0.1 127.0.0.1 - - 1592327077547024 493 "GET /wordpress/wp-content/themes/twentyseventeen/style.css?ver=20190507 HTTP/1.1" 200 84284 80 54132 "http://localh
127.0.0.1 127.0.0.1 - - 1592327077547606 174 "GET /wordpress/wp-content/themes/twentyseventeen/assets/css/blocks.css?ver=20190105 HTTP/1.1" 200 10516 80 54132 "h
127.0.0.1 127.0.0.1 - - 1592327077547870 170 "GET /wordpress/wp-content/themes/twentyseventeen/assets/css/colors-dark.css?ver=20190408 HTTP/1.1" 200 18542 80 541
127.0.0.1 127.0.0.1 - - 1592327077548124 572 "GET /wordpress/wp-includes/js/jquery/jquery.js?ver=1.12.4-wp HTTP/1.1" 200 97197 80 54132 "http://localhost/wordpre
127.0.0.1 127.0.0.1 - - 1592327077548815 164 "GET /wordpress/wp-includes/js/jquery/jquery-migrate.min.js?ver=1.4.1 HTTP/1.1" 200 10379 80 54132 "http://localhost
127.0.0.1 127.0.0.1 - - 1592327077549061 40205 "GET /wordpress/index.php/wp-json/ HTTP/1.1" 200 84854 80 54132 "http://localhost/wordpress/index.php/blog/" "Wget
127.0.0.1 127.0.0.1 - - 1592327077589372 29088 "GET /wordpress/xmlrpc.php?rsd HTTP/1.1" 200 1058 80 54132 "http://localhost/wordpress/index.php/blog/" "Wget/1.19
127.0.0.1 127.0.0.1 - - 1592327077618742 192 "GET /wordpress/wp-includes/wlwmanifest.xml HTTP/1.1" 200 1359 80 54132 "http://localhost/wordpress/index.php/blog/"
127.0.0.1 127.0.0.1 - - 1592327077619019 1861 "GET /wordpress/wp-content/uploads/2020/04/100_8745.png HTTP/1.1" 200 563990 80 54132 "http://localhost/wordpress/i
127.0.0.1 127.0.0.1 - - 1592327077621195 56465 "GET /wordpress/ HTTP/1.1" 200 68697 80 54132 "http://localhost/wordpress/index.php/blog/" "Wget/1.19.4 (linux-gnu
127.0.0.1 127.0.0.1 - - 1592327077677770 57796 "GET /wordpress/index.php/tribunal/ HTTP/1.1" 200 60473 80 54132 "http://localhost/wordpress/index.php/blog/" "Wge
127.0.0.1 127.0.0.1 - - 1592327077735685 46155 "GET /wordpress/index.php/publicaciones/ HTTP/1.1" 200 60009 80 54132 "http://localhost/wordpress/index.php/blog/"
127.0.0.1 127.0.0.1 - - 1592327077781929 44418 "GET /wordpress/index.php/acerca-de/ HTTP/1.1" 200 58175 80 54132 "http://localhost/wordpress/index.php/blog/" "Wg
127.0.0.1 127.0.0.1 - - 1592327077826456 43000 "GET /wordpress/index.php/contacto/ HTTP/1.1" 200 58070 80 54132 "http://localhost/wordpress/index.php/blog/" "Wge
127.0.0.1 127.0.0.1 - - 1592327077869559 70631 "GET /wordpress/index.php/2020/06/01/metodos-get-y-post-de-http/ HTTP/1.1" 200 67153 80 54132 "http://localhost/wo
127.0.0.1 127.0.0.1 - - 1592327077940313 246 "GET /wordpress/wp-content/uploads/2020/06/http_GET.jpg HTTP/1.1" 200 15137 80 54132 "http://localhost/wordpress/ind
127.0.0.1 127.0.0.1 - - 1592327077940686 88344 "GET /wordpress/index.php/2020/04/12/propuestadeproyecto/ HTTP/1.1" 200 67713 80 54132 "http://localhost/wordpress
127.0.0.1 127.0.0.1 - - 1592327078029726 225 "GET /wordpress/wp-content/themes/twentyseventeen/assets/js/skip-link-focus-fix.js?ver=20161114 HTTP/1.1" 200 1003 8
127.0.0.1 127.0.0.1 - - 1592327078030694 191 "GET /wordpress/wp-content/themes/twentyseventeen/assets/js/navigation.js?ver=20161203 HTTP/1.1" 200 4075 80 54132 "
127.0.0.1 127.0.0.1 - - 1592327078031685 78 "GET /wordpress/wp-content/themes/twentyseventeen/assets/js/global.js?ver=20190121 HTTP/1.1" 200 8076 80 54132 "http:
127.0.0.1 127.0.0.1 - - 1592327078031940 80 "GET /wordpress/wp-content/themes/twentyseventeen/assets/js/jquery.scrollTo.js?ver=2.1.2 HTTP/1.1" 200 6158 80 54132
127.0.0.1 127.0.0.1 - - 1592327078032662 82 "GET /wordpress/wp-includes/js/wp-embed.min.js?ver=5.4.2 HTTP/1.1" 200 1755 80 54132 "http://localhost/wordpress/inde
```

Figura 5-1: Registros del servidor al realizar una petición GET

Ahora que tenemos escogida la manera de generar peticiones, el siguiente paso a realizar es un estudio sobre cómo enviar estas peticiones cada cierto tiempo, ya que no tiene sentido enviar muchas peticiones a la vez y en un breve intervalo de tiempo. Como estamos buscando que el tráfico generado sea lo más realista posible, normalmente los clientes cuando acceden a un servidor no lo hacen todos a la vez, sino que cada uno se mete en momentos diferentes y, cabe destacar, que el comportamiento humano a la hora de pinchar sobre un enlace sigue una distribución de Poisson, pero una vez que han pinchado sobre el enlace, las descargas que se generan no siguen esta distribución, lo que hace que finalmente tienda a una distribución de Pareto. Para simular este tipo de situaciones, se hará que el tiempo de llegada de las primeras peticiones sigan dicha distribución de Poisson. Este tipo de distribución de probabilidad discreta se utiliza en situaciones en las que se quiere determinar el número de eventos que ocurren en un intervalo de tiempo. Para comprobar que realmente se sigue esta distribución, en la siguiente figura podemos observar tres gráficas: su CDF, su CDF inversa y su CDF inversa con el eje logarítmico.

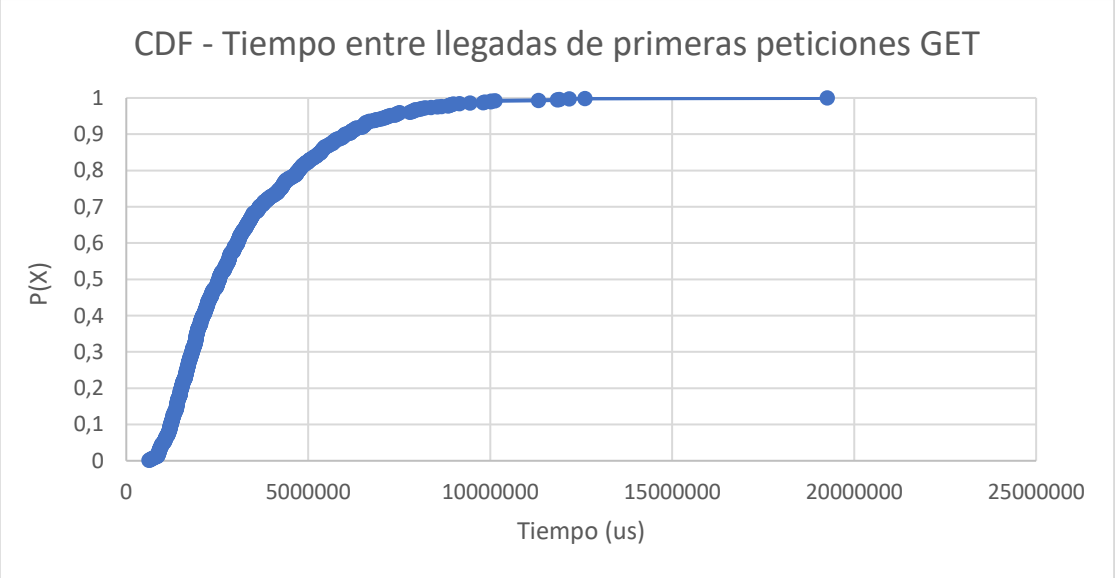

Figura 5-2: CDF del tiempo entre llegadas de primeras peticiones GET

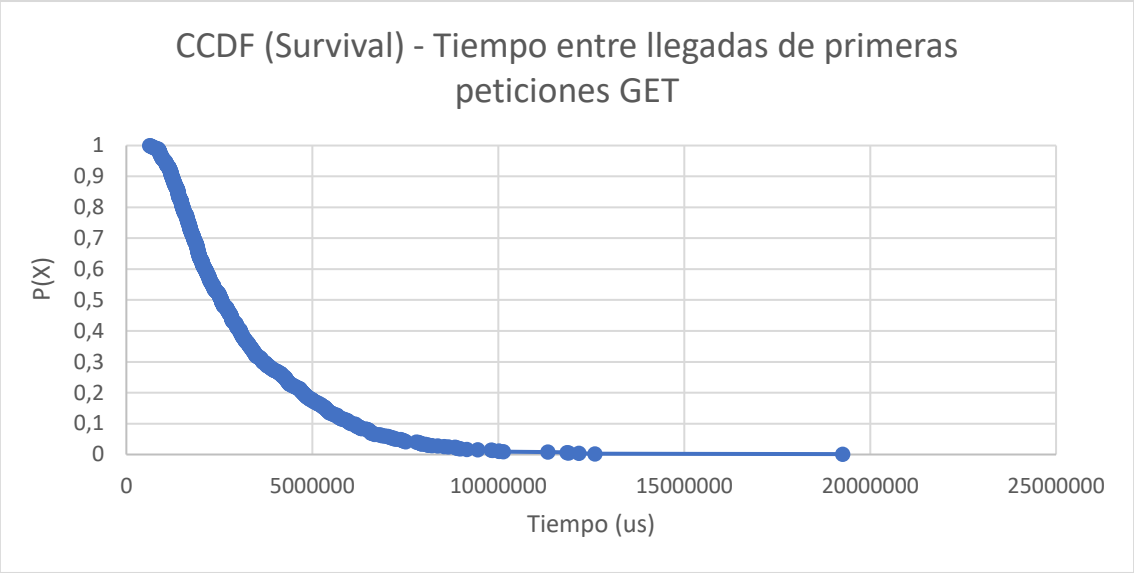

Figura 5-3: CDF inversa del tiempo entre llegadas de primeras peticiones GET

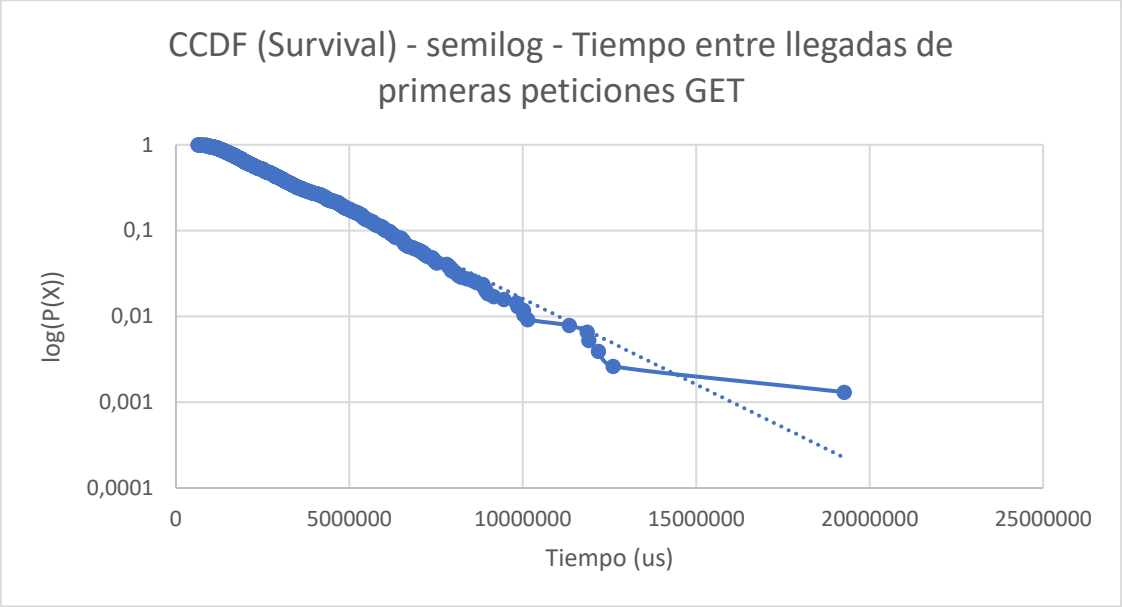

Figura 5-4: CDF inversa con eje y logarítmico del tiempo entre llegadas de primeras peticiones GET

Con estos métodos aplicados podemos generar tráfico con el script sobre ambos servidores, pero todavía no es un tráfico muy real, ya que solo se realizan peticiones a la página web principal del servidor.

La siguiente mejora añadida ha sido incluir en el script varias páginas posibles diferentes del servidor, concretamente se han añadido ocho posibilidades de páginas web. Con esta mejora, nuestro tráfico generado se va pareciendo más a uno real. Pero, aun así, siguen faltando características a añadir. En un tráfico real el cliente puede generar peticiones erróneas o realizar peticiones de objetos que el servidor no contenga, por lo que también

se han tenido en cuenta estos casos. Se han añadido peticiones erróneas y peticiones con objetos que no contiene el servidor.

Con esta mejora incluida, ya tenemos completado nuestro script que generará tráfico sobre los servidores HTTP y HTTPS, pero hay que remarcar que ahora con el tráfico que se genera no se producen pérdidas de paquetes, retardos, etc. Por ello, a través de la herramienta *Traffic Control* [32] que incorpora Linux, simularemos las pérdidas de paquetes, retardos y limitaciones de ancho de banda sobre la red.

En la Tabla 5-2 se muestra una lista indicando las bases de datos generadas y qué tipo de información contienen.

Tabla 5-2: Lista de bases de datos generadas

<table>
<tr><th>Bases de datos generadas</th><th>Servidor HTTP</th><th>Servidor HTTPS</th></tr>
<tr><td>Tráfico Ideal</td><td rowspan="4">Log de acceso del servidor

Captura del tráfico con Wireshark, incluyendo el plugin TRANSUM

Log de HTTP y TCP que genera TSTAT</td><td rowspan="4">Log de acceso del servidor

Captura del tráfico con Wireshark</td></tr>
<tr><td>Tráfico con peticiones erróneas</td></tr>
<tr><td>Tráfico con retardos (Traffic Control)</td></tr>
<tr><td>Tráfico con pérdidas de paquetes (Traffic Control)</td></tr>
<tr><td>Tráfico con retardos y limitación en ancho de banda (Traffic Control)</td><td></td><td></td></tr>
</table>

Por último, hay que indicar que el script generado se deja ejecutándose durante varias horas para así tener una traza grande (de varios MBytes) tanto del *log* de acceso de los servidores como de las capturas realizadas mediante Wireshark y, en el caso de HTTP, la traza procesada con la herramienta TSTAT. A estas trazas se le aplicarán los algoritmos que se desarrollan en la siguiente sección.

5.3. Descripción de los Algoritmos

El primer paso que hay que realizar tras obtener el *log* de acceso del servidor y su traza de red asociada es realizar un filtrado sobre ambos ficheros, para cada servidor, primero procesamos el *log* de acceso para quedarnos con la información que nos resulta más importante para desarrollar el algoritmo. Nos quedamos con las direcciones IP del cliente y servidor, así como sus puertos asociados; la primera línea de la petición; el estado de la respuesta del servidor; el tiempo en el que llegó la petición; y, por último, el tiempo que tarda el servidor en servir la respuesta. En cuanto al servidor HTTP, hacemos el mismo filtro sobre el fichero de *log* HTTP que genera la herramienta TSTAT, y respecto a la herramienta TRANSUM que contiene Wireshark, no es necesario realizar ningún filtrado ya que no contiene información adicional que necesitemos descartar.

Una vez que tenemos los datos filtrados, podemos empezar el desarrollo *software* de este Trabajo de Fin de Máster. Este desarrollo se ha realizado en el lenguaje de programación AWK, se pensó inicialmente utilizar el lenguaje C o Python, pero AWK es idóneo para trabajar sobre grandes ficheros y nos permite filtrar los datos por columnas. Se decidió finalmente utilizar este lenguaje por la estructura que tienen los ficheros de *log* y las capturas, todos estos ficheros tienen los datos organizados por columnas, por ello, para que el algoritmo tenga buen rendimiento y sea rápido en procesar dichos ficheros tan grandes se optó por AWK, dado que proporciona mejor rendimiento que C y Python a la hora de procesar ficheros por columnas. En primer lugar, se desarrollará el algoritmo para el servidor HTTP, posteriormente, este algoritmo se adaptará para que funcione sobre el servidor HTTPS.

5.3.1. Algoritmo para la correlación de registros en el servidor HTTP

En este apartado se explica el desarrollo del algoritmo para correlar registros en un servidor HTTP. Se ha diseñado el algoritmo partiendo del caso ideal, esto es, se ha partido de la base de datos creada con las trazas de red y los *log* de acceso en el caso ideal, en el que no se producen pérdidas de paquetes, retardos y ni se producen limitaciones de ancho de banda. A partir de aquí se han realizado dos adaptaciones del algoritmo, una para las bases de datos generadas con la herramienta *Traffic Control*, y la otra, dividimos el tráfico en objetos estáticos y dinámicos. A continuación, comenzamos con la explicación de la implementación del algoritmo.

Primero, cogemos el fichero de *log* HTTP que genera la herramienta TSTAT. Como podemos ver en la Figura 5-5, en este *log* se van intercalando las peticiones y respuestas realizadas, esto es, en la primera línea se muestra la primera petición generada y en la siguiente la respuesta realizada por el servidor, y así con el resto de las peticiones y respuestas.

```
127.0.0.1   54132   127.0.0.1   80  1592327077.007184   GET localhost   -   /wordpress/index.php/blog/  -   Wget/1.19.4 (linux-gnu)
127.0.0.1   54132   127.0.0.1   80  1592327077.448439   HTTP    200 -   text/html; charset=UTF-8    Apache/2.4.29 (Ubuntu)  -   -
127.0.0.1   54132   127.0.0.1   80  1592327077.466601   GET localhost   -   /wordpress/index.php/feed/  http://localhost/wordpress/.
127.0.0.1   54132   127.0.0.1   80  1592327077.505480   HTTP    200 -   application/rss+xml; charset=UTF-8  Apache/2.4.29 (Ubuntu)
```

Figura 5-5: Contenido del *log* HTTP de la herramienta TSTAT

Al generarse directamente de esta manera, la implementación de este algoritmo ha sido mucho más sencilla ya que no aparecen las peticiones y respuestas desordenadas. Ahora, para calcular el tiempo de respuesta que observa el cliente, se ha ido realizando la diferencia del tiempo de la respuesta junto con el tiempo del envío de su petición correspondiente, y dicho resultado se incluye en un fichero que contendrá los tiempos de respuesta, las IPs de cliente y servidor, y los puertos de cada uno.

El siguiente paso realizado ha sido añadir a este fichero generado los tiempos de respuesta, la información de las IPs y de los puertos que registra el *log* de acceso del servidor y el plugin TRANSUM de Wireshark.

Posteriormente, se ha adaptado el algoritmo para el caso en el que el tráfico se ha generado con la herramienta *Traffic Control* (pérdidas de paquetes, retardos y cuando se limita el ancho de banda). Esta adaptación es debido a que se producen retransmisiones en los flujos entre el cliente y el servidor, por lo que tendremos, en algunas ocasiones, varias peticiones y varias respuestas. A continuación, en la siguiente figura, podemos ver un ejemplo de estos casos:

```
127.0.0.1  55766  127.0.0.1  80  1595527511.062107  HTTP  200  563701 image/png
127.0.0.1  55766  127.0.0.1  80  1595527515.579332  HTTP  200  563701 image/png
127.0.0.1  55766  127.0.0.1  80  1595527515.585179  GET localhost  -  /wordpress/
127.0.0.1  55766  127.0.0.1  80  1595527515.638889  GET localhost  -  /wordpress/
127.0.0.1  55768  127.0.0.1  80  1595527516.740850  GET localhost  -  /wordpress/
```

Figura 5-6: Ejemplo de retransmisión de peticiones y respuestas HTTP

En estas situaciones, para calcular el tiempo de respuesta, lo que se ha diseñado es que el algoritmo tenga en cuenta el tiempo de la primera petición descartando el resto que se retransmiten, ya que la primera petición es la que genera el cliente al solicitar un objeto, y las peticiones que se retransmiten se deben a que el servidor está tardando demasiado en gestionar dicha petición. En cuanto a las respuestas, se ha cogido el tiempo de la última que envía el servidor, ya que con ésta es cuando el servidor ha entregado toda la información que ha solicitado el cliente.

Para todas las bases de datos generadas, se ha realizado la división entre objetos estáticos y dinámicos para realizar una correlación más detallada. Para ello, hemos tenido que analizar qué objetos contiene la URL que se registra en la petición, y a su vez hemos asociado dichas peticiones con sus respuestas para los tiempos de respuesta de cada caso.

Por último, este algoritmo diseñado nos generará para cada base de datos los siguientes ficheros:

- Un fichero con los tiempos de respuesta entre cada petición y su respuesta asociada para cada flujo,
- Un fichero con los tiempos de respuesta de las peticiones a objetos estáticos y su respuesta asociada para cada flujo,
- Y un fichero con los tiempos de respuesta de las peticiones a objetos dinámicos y su respuesta asociada para cada flujo.

Una vez que tenemos todos los ficheros con los tiempos de respuesta calculados, podremos comparar gráfica y estadísticamente dichos resultados.

5.3.2. Algoritmo para la correlación de registros en el servidor HTTPS

En este apartado se explica el desarrollo del algoritmo para correlar registros en un servidor HTTPS. En el caso de este servidor solamente se han utilizado el *log* de acceso del servidor y la captura realizada con Wireshark, en el Capítulo 6 explicaremos por qué no se ha utilizado la herramienta TSTAT.

El primer paso realizado ha sido desarrollar el algoritmo partiendo del diseñado para el servidor HTTP y se han seguido pasos similares comenzando su desarrollo con la base de datos de tráfico ideal. Respecto a la parte realizada con el *log* de acceso se deja igual, ya que, aunque el servidor esté cifrado, su *log* de acceso no lo está y se pueden observar todas las peticiones recibidas sin problema. En cuanto a la parte cifrada, podemos ver en la Figura 5-7 cómo es este tipo de tráfico, el cliente comienza la negociación explicada en el Capítulo 2, y una vez terminada comienza el intercambio de mensajes. Para calcular los tiempos de respuesta se va a obviar este paso de negociación, ya que como se comentó en [1] es una tarea trivial y fácil de calcular, por lo que se cogerá la primera petición (es el correspondiente al primer mensaje de *Application Data*) que realiza el cliente justo después de terminar esta negociación.

Time	Source	Destination	Protocol	Length	SrcPort	DstPort	Info
1594746651.105171922	127.0.0.1	127.0.0.1	TLSv1.3	377	35298	443	Client Hello
1594746651.105218291	127.0.0.1	127.0.0.1	TCP	66	443	35298	443 → 35298 [ACK] Seq=1 Ack=312 Win=6
1594746651.109494797	127.0.0.1	127.0.0.1	TLSv1.3	1563	443	35298	Server Hello, Change Cipher Spec, App
1594746651.109511258	127.0.0.1	127.0.0.1	TCP	66	35298	443	35298 → 443 [ACK] Seq=312 Ack=1498 Wi
1594746651.110399228	127.0.0.1	127.0.0.1	TLSv1.3	146	35298	443	Change Cipher Spec, Application Data
1594746651.111226763	127.0.0.1	127.0.0.1	TCP	66	443	35298	443 → 35298 [ACK] Seq=1498 Ack=392 Wi
1594746651.111233266	127.0.0.1	127.0.0.1	TLSv1.3	275	35298	443	Application Data
1594746651.111238133	127.0.0.1	127.0.0.1	TCP	66	443	35298	443 → 35298 [ACK] Seq=1498 Ack=601 Wi
1594746651.112152059	127.0.0.1	127.0.0.1	TLSv1.3	337	443	35298	Application Data
1594746651.112166624	127.0.0.1	127.0.0.1	TCP	66	35298	443	35298 → 443 [ACK] Seq=601 Ack=1769 Wi
1594746651.112410226	127.0.0.1	127.0.0.1	TLSv1.3	337	443	35298	Application Data
1594746651.112414786	127.0.0.1	127.0.0.1	TCP	66	35298	443	35298 → 443 [ACK] Seq=601 Ack=2040 Wi

Figura 5-7: Negociación en tres pasos de tráfico HTTPS

En este caso el cálculo de los tiempos de respuesta también ha sido relativamente sencillo de calcular, ya que en general el cliente siempre manda un solo mensaje de *Application Data*, pero en ocasiones el servidor envía la respuesta en dos o más mensajes. En los casos en los que ocurre esta situación, lo que se ha hecho es coger el tiempo del último mensaje enviado por el servidor y realizando la diferencia de este tiempo con el tiempo de envío de la petición del cliente, así obtenemos el tiempo de respuesta que percibe el cliente. Con esto, ya tenemos generado nuestro fichero que contendrá la información de los tiempos

de respuesta obtenidos del *log* de acceso junto con los tiempos de respuesta observados en la captura del tráfico.

Para este algoritmo, no solamente se han calculado los tiempos entre cada petición y respuesta, sino que también se han calculado los tiempos entre cada petición realizada de cada flujo y el tiempo entre la primera petición y la última respuesta de cada flujo. Este último cálculo nos es de gran utilidad en las situaciones en las que se han generado peticiones erróneas, es decir, cuando el cliente ha solicitado un objeto que no se encuentra en el servidor. El *log* de acceso registra esta petición indicando el tiempo que tarda en servirla, pero lo que se ve en la red es distinto, ya que no hay manera de saber el estado de cada respuesta. Como lo que se está haciendo es una diferencia entre la primera petición y la última respuesta del flujo, en estos casos el *log* de acceso solo registra una petición y la diferencia resultante que se obtendrá será de cero, con ello seremos capaces de identificar dichas peticiones fácilmente.

Posteriormente, se ha adaptado el algoritmo, al igual que se hizo para el servidor HTTP, para el caso en el que el tráfico se ha generado con la herramienta *Traffic Control* (pérdidas de paquetes, retardos y cuando se limita el ancho de banda). Ahora ocurre la misma situación que en HTTP, pero a través de los mensajes *Application Data*, en las situaciones en las que haya más de una petición se cogerá el tiempo de envío de la primera que realice el cliente, y cuando el servidor envíe más de un mensaje *Application Data* se cogerá el tiempo de la última que envíe. Con estos tiempos realizamos la diferencia y obtendremos el tiempo de respuesta que percibe el cliente.

También hay que indicar, que en este servidor no es posible realizar la división entre objetos estáticos y dinámicos de la captura de tráfico realizada. Podemos sacar esa información del *log* de acceso del propio servidor, pero no podemos compararlo con la captura realizada al estar la traza cifrada.

Por último, este algoritmo diseñado nos generará para cada base de datos los siguientes ficheros:

- Un fichero con los tiempos de respuesta entre las primeras peticiones de cada flujo,
- Un fichero con los tiempos de respuesta entre la primera petición y la última respuesta de cada flujo,
- Y un fichero con los tiempos de respuesta entre cada petición y respuesta asociada de cada flujo.

Con estos tres ficheros generados podremos comparar gráfica y estadísticamente dichos resultados.

5.4. Conclusiones

Esta fase de desarrollo del algoritmo se ha realizado junto con la fase de pruebas para comprobar los resultados y realizar las modificaciones necesarias.

Ha sido una fase fundamental en el proyecto, ya que sin una buena base de datos generada y un buen desarrollo del algoritmo no se conseguiría analizar de una manera correcta y real los tiempos de respuesta del *log* de acceso del servidor y de las capturas realizadas.

Por último, se han ido realizando múltiples modificaciones y mejoras a los algoritmos para conseguir los resultados deseados y mejores tiempos de ejecución.

6. Validación y Resultados

6.1. Introducción

En este capítulo se mostrarán las diferentes pruebas realizadas para este Trabajo de Fin de Máster. Primero, evaluaremos el algoritmo aplicado sobre el servidor HTTP y, posteriormente, evaluaremos el algoritmo aplicado sobre el servidor HTTPS. También se comentarán los resultados más relevantes observados durante las pruebas.

Por último, podremos ver cómo de buena es la información que provee el *log* de acceso de los servidores frente a los datos que se observan en la red.

6.2. Evaluación del Servidor HTTP

Las pruebas realizadas con el servidor HTTP han sido las de aplicar el algoritmo y sus adaptaciones a las bases de datos generadas. A continuación, mostraremos los resultados sobre todas las bases de datos:

- Resultados sobre la base de datos: Tráfico Ideal

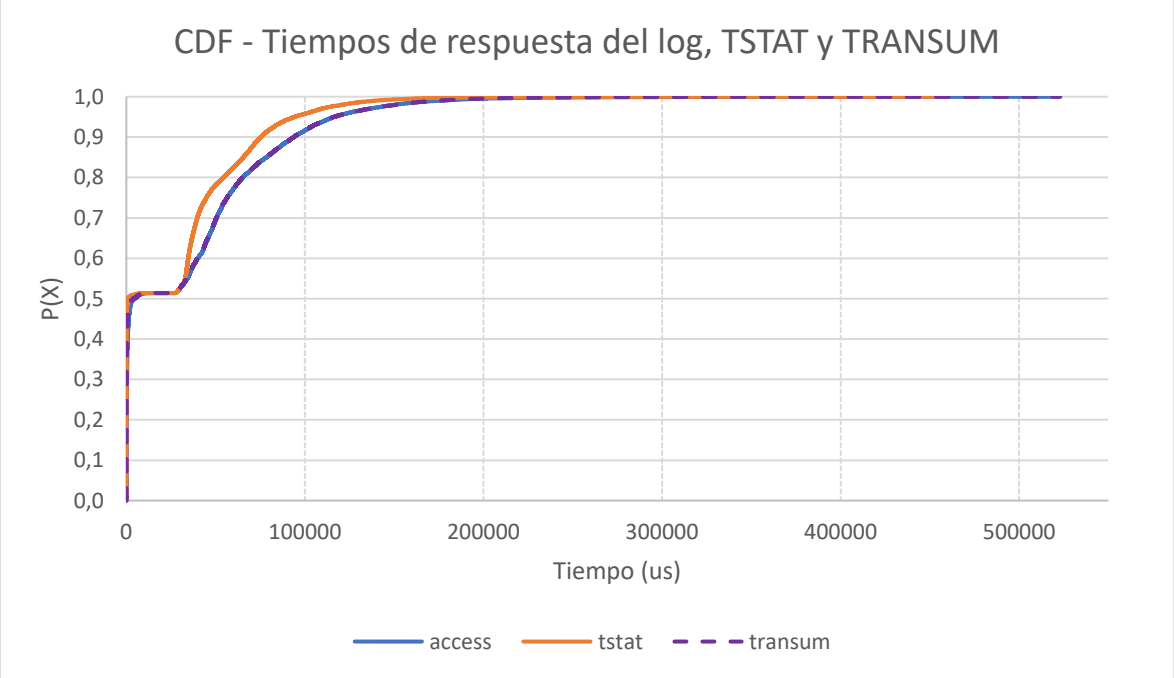

Figura 6-1: CDF de los tiempos de respuesta del servidor HTTP entre peticiones del *log* de acceso, TSTAT y TRANSUM

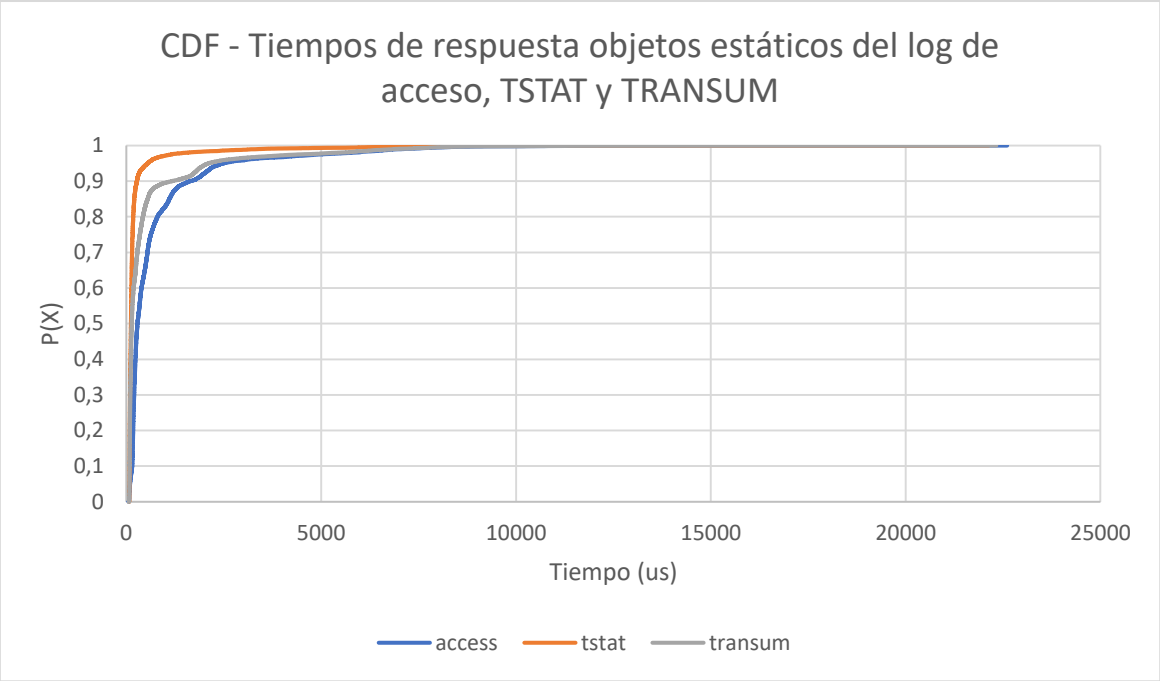

Figura 6-2: CDF de los tiempos de respuesta del servidor HTTP entre peticiones de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

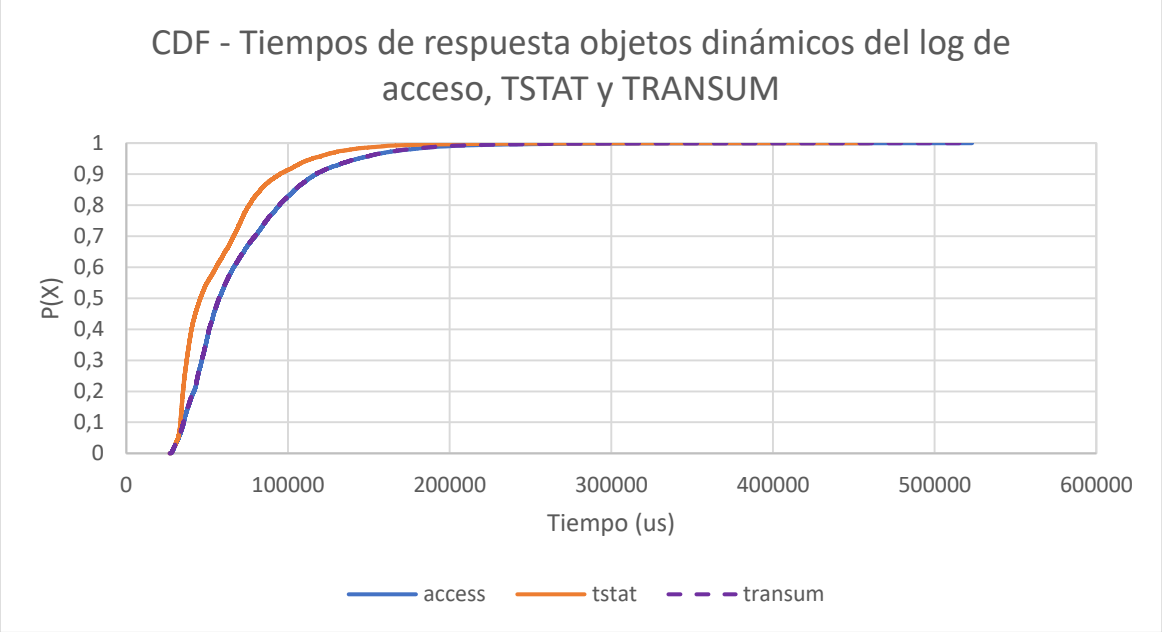

Figura 6-3: CDF de los tiempos de respuesta del servidor HTTP entre peticiones de objetos dinámicos del *log* de acceso, TSTAT y TRANSUM

En estas tres gráficas podemos ver que los tiempos de respuesta obtenidos con la herramienta TRANSUM son prácticamente los mismos que registra el *log* de acceso. La diferencia principal se observa en las peticiones estáticas, para las cuales los tiempos de respuesta son menores respecto a los tiempos de respuesta de las peticiones dinámicas, esto es debido a que el servidor procesa en menos tiempo los objetos estáticos al ocupar menos bytes. En la Figura 6-1 podemos observar que alrededor del 50% de las peticiones

son estáticas y el restante son dinámicas, de ahí que se observe ese cambio en los tiempos de respuesta.

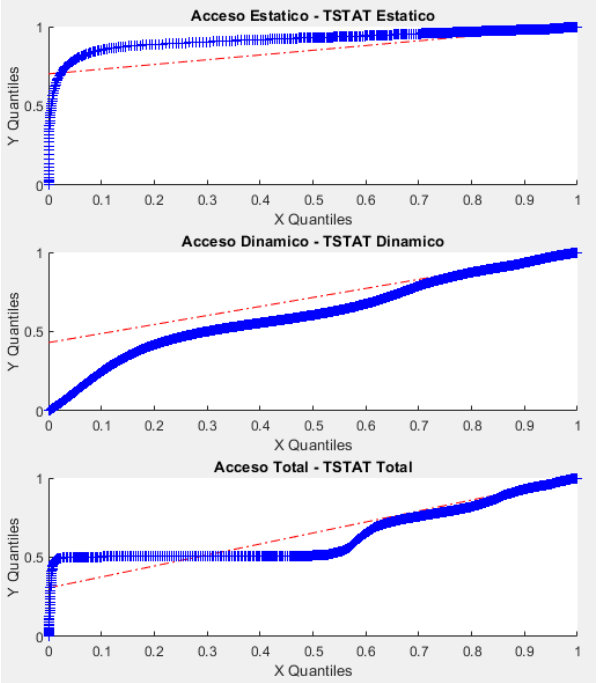

Figura 6-4: qqplots sobre el *log* de acceso del servidor HTTP frente al *log* HTTP de TSTAT

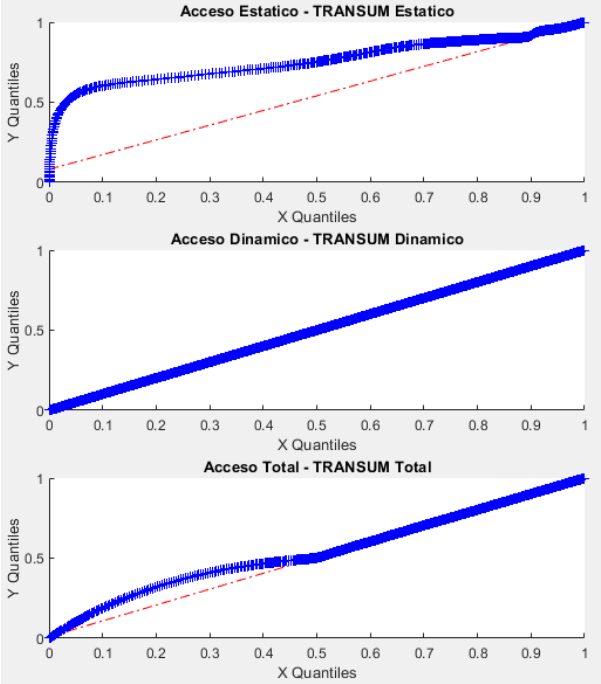

Figura 6-5: qqplots sobre el *log* de acceso del servidor HTTP frente al TRANSUM de Wireshark

En estas dos gráficas podemos observar cuánto se parecen las distribuciones de los tiempos de respuesta obtenidos con las herramientas TSTAT y TRANSUM frente a la distribución que siguen los tiempos del *log* de acceso. En la Figura 6-4 podemos observar

que los tiempos que obtiene el TSTAT no siguen la distribución de los tiempos del *log* de acceso, mientras que para el TRANSUM podemos ver que la distribución dinámica sigue exactamente la misma que la dinámica del *log* de acceso. Y también podemos observar en la tercera gráfica de la Figura 6-5 lo mencionado anteriormente en las CDFs, el 50% de la gráfica en la que se producen peticiones estáticas no sigue la distribución del *log* de acceso, pero el resto en el que se producen las dinámicas si coincide con la del *log* de acceso.

Tabla 6-1: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el *log* HTTP de TSTAT

Métodos estadísticos	*Log* de acceso total vs TSTAT total	*Log* de acceso estático vs TSTAT estático	*Log* de acceso dinámico vs TSTAT dinámico
Test KS	1	1	1
Divergencia KL	0,0018	0,0062	0,0029

Tabla 6-2: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el TRANSUM de Wireshark

Métodos estadísticos	*Log* de acceso total vs TRANSUM total	*Log* de acceso estático vs TRANSUM estático	*Log* de acceso dinámico vs TRANSUM dinámico
Test KS	1	1	0
Divergencia KL	1,1246e-4	0,0032	1,4525e-7

Con estas medidas realizadas para cada herramienta, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado una hipótesis nula en el caso de los objetos dinámicos del TRANSUM frente al *log* de acceso, esto quiere decir que ambas distribuciones son prácticamente idénticas, mientras que para el resto da como resultado el rechazo de la hipótesis nula. Y respecto a la divergencia KL podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con peticiones erróneas

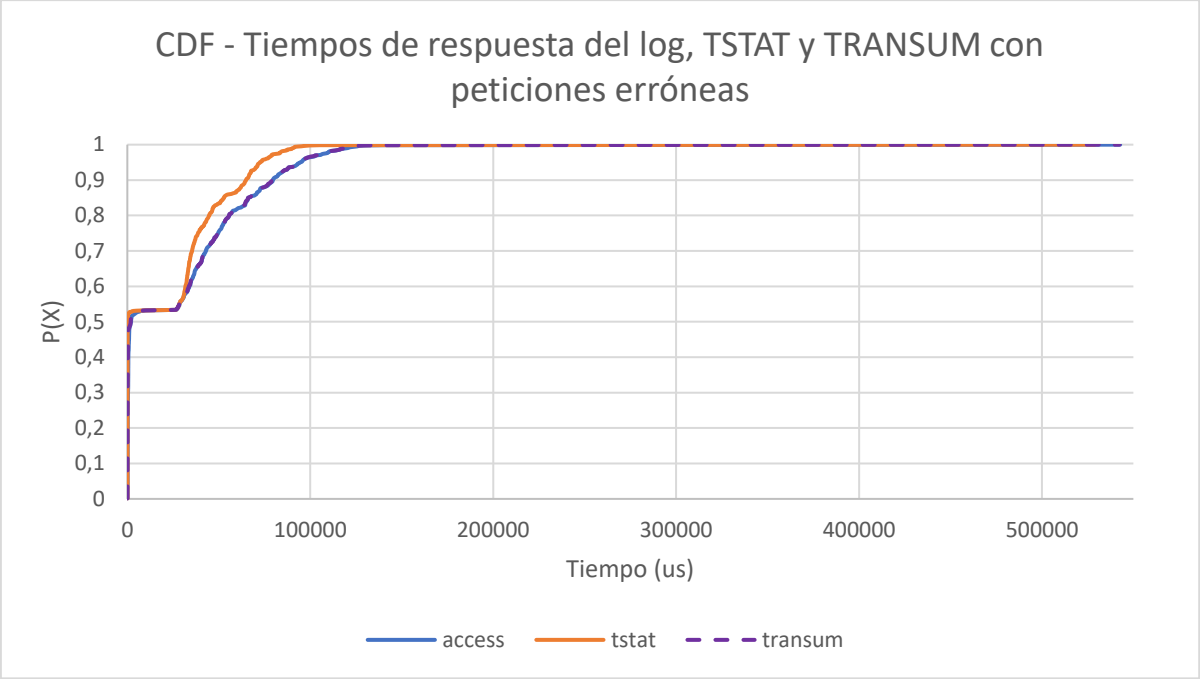

Figura 6-6: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (incluyendo peticiones erróneas) del *log* de acceso, TSTAT y TRANSUM

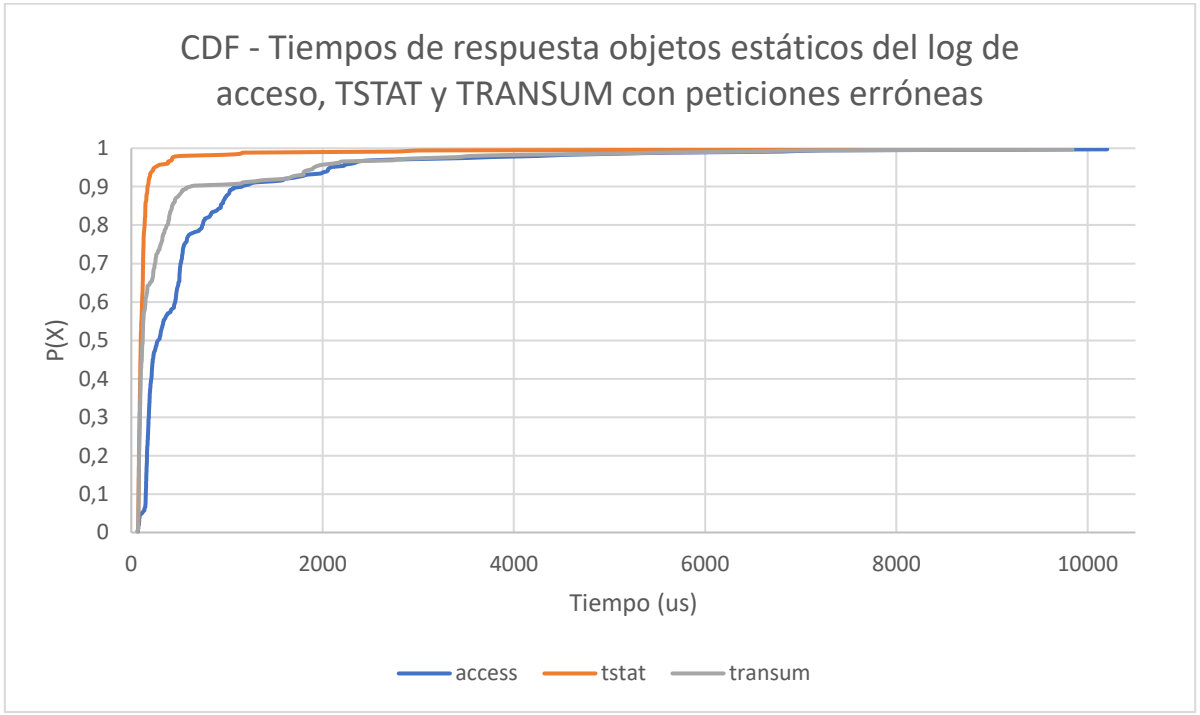

Figura 6-7: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (incluyendo peticiones erróneas) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

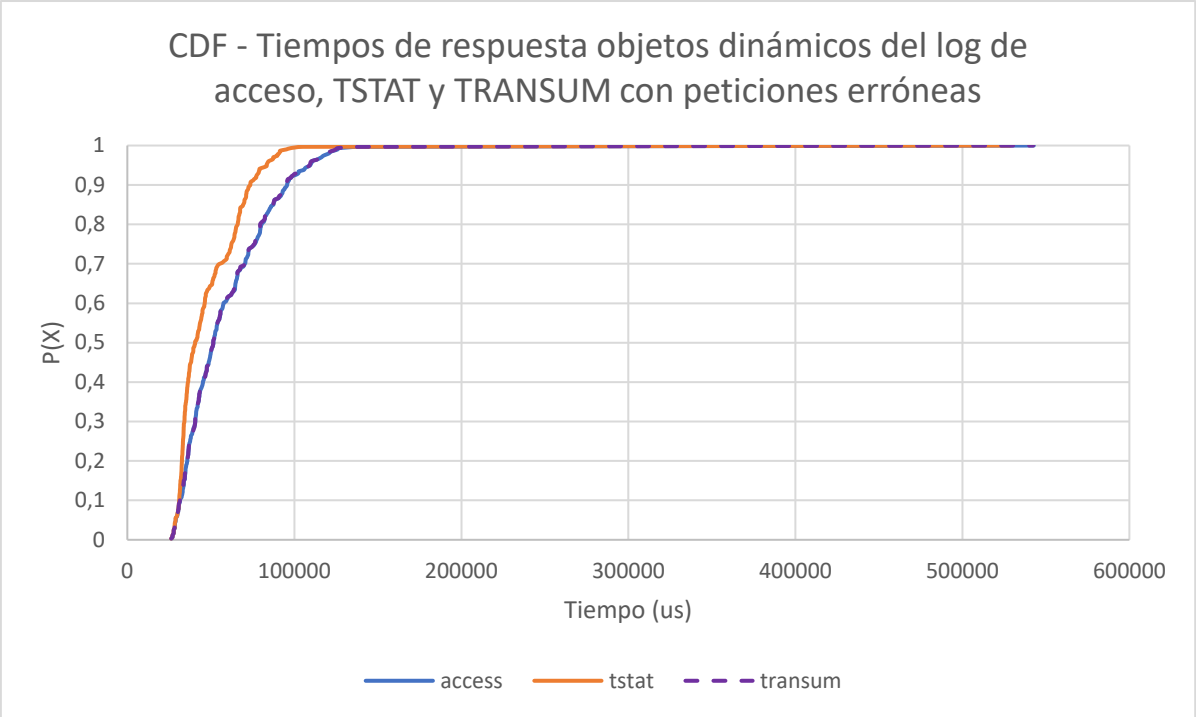

Figura 6-8: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (incluyendo peticiones erróneas) de objetos dinámicos del *log* de acceso, TSTAT y TRANSUM

En esta base de datos ocurre algo parecido a la del tráfico ideal, la parte dinámica del TRANSUM coincide con la del *log* de acceso mientras que la estática difiere bastante. En cuanto al TSTAT sucede lo mismo que anteriormente y se observa cómo no sigue la distribución de los tiempos del *log* de acceso. También se observa que alrededor del 50% del tiempo se producen peticiones estáticas y el restante dinámicas.

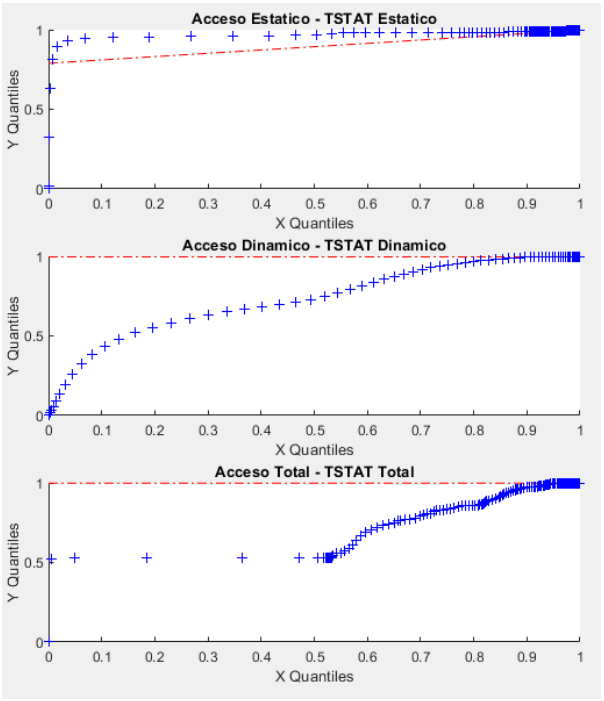

Figura 6-9: qqplots sobre el *log* de acceso del servidor HTTP frente al *log* HTTP de TSTAT (incluyendo peticiones erróneas)

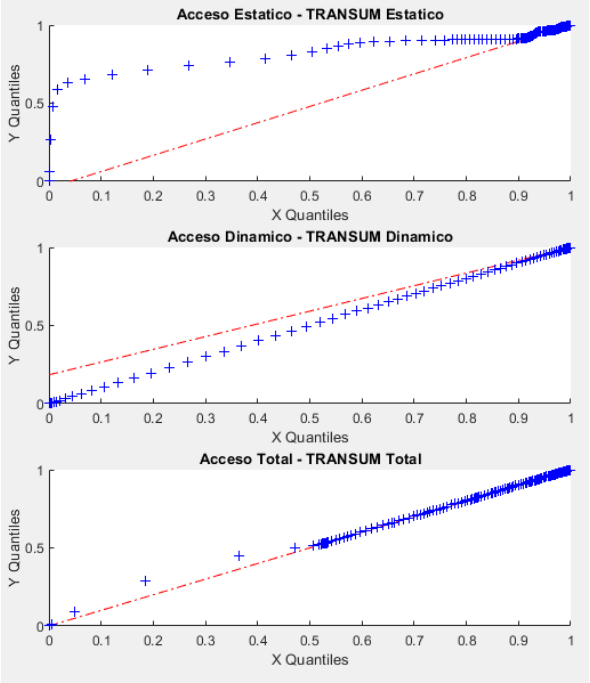

Figura 6-10: qqplots sobre el *log* de acceso del servidor HTTP frente al TRANSUM de Wireshark (incluyendo peticiones erróneas)

Con estas dos gráficas, podemos observar que ocurre la misma situación que en la base de datos de tráfico ideal. La herramienta TRANSUM es la que se parece más a la distribución de los tiempos del *log* de acceso.

Tabla 6-3: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el *log* HTTP de TSTAT (incluyendo peticiones erróneas)

Métodos estadísticos	*Log* de acceso total vs TSTAT total	*Log* de acceso estático vs TSTAT estático	*Log* de acceso dinámico vs TSTAT dinámico
Test KS	1	1	1
Divergencia KL	0,0020	0,0205	0,0093

Tabla 6-4: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el TRANSUM de Wireshark (incluyendo peticiones erróneas)

Métodos estadísticos	*Log* de acceso total vs TRANSUM total	*Log* de acceso estático vs TRANSUM estático	*Log* de acceso dinámico vs TRANSUM dinámico
Test KS	0	1	0
Divergencia KL	7,1447e-5	0,0116	2,4263e-7

Con estas medidas realizadas para cada herramienta, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado una hipótesis nula en el caso de los objetos dinámicos y la captura total del TRANSUM frente al *log* de acceso, esto quiere decir que ambas distribuciones son prácticamente idénticas, mientras que para el resto da como resultado el rechazo de la hipótesis nula. Y respecto a la divergencia KL podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con retardos

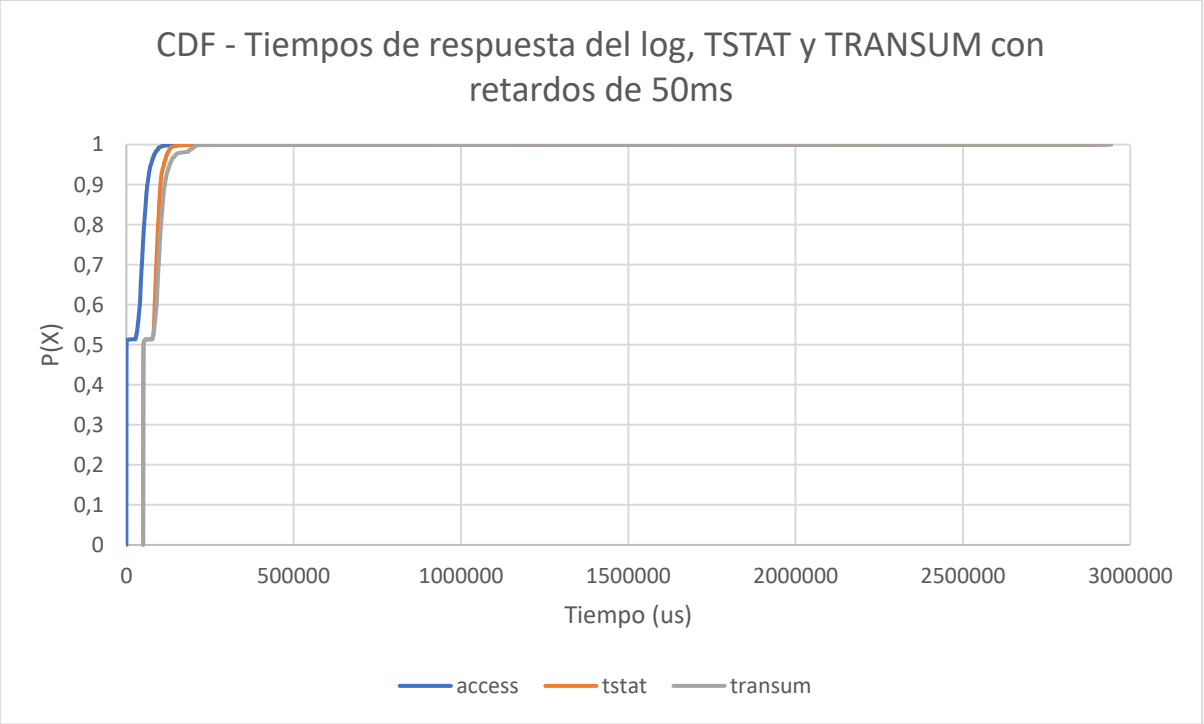

Figura 6-11: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con retardos de 50ms) del *log* de acceso, TSTAT y TRANSUM

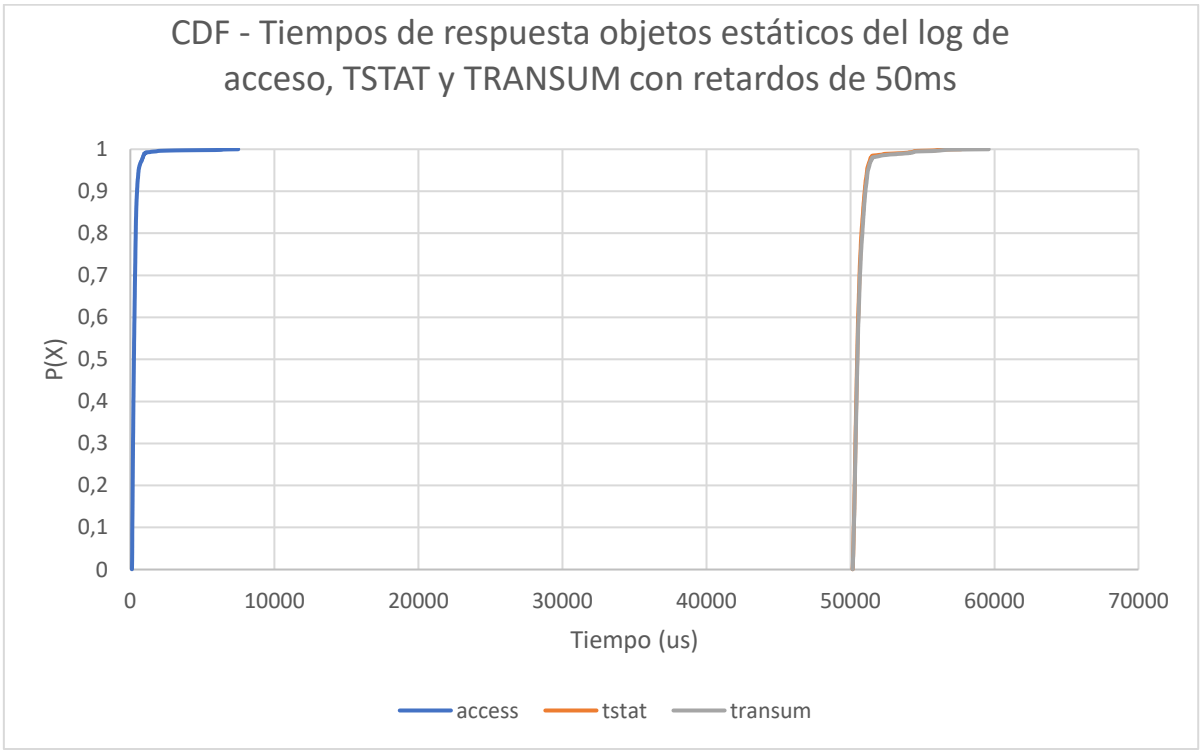

Figura 6-12: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con retardos de 50ms) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

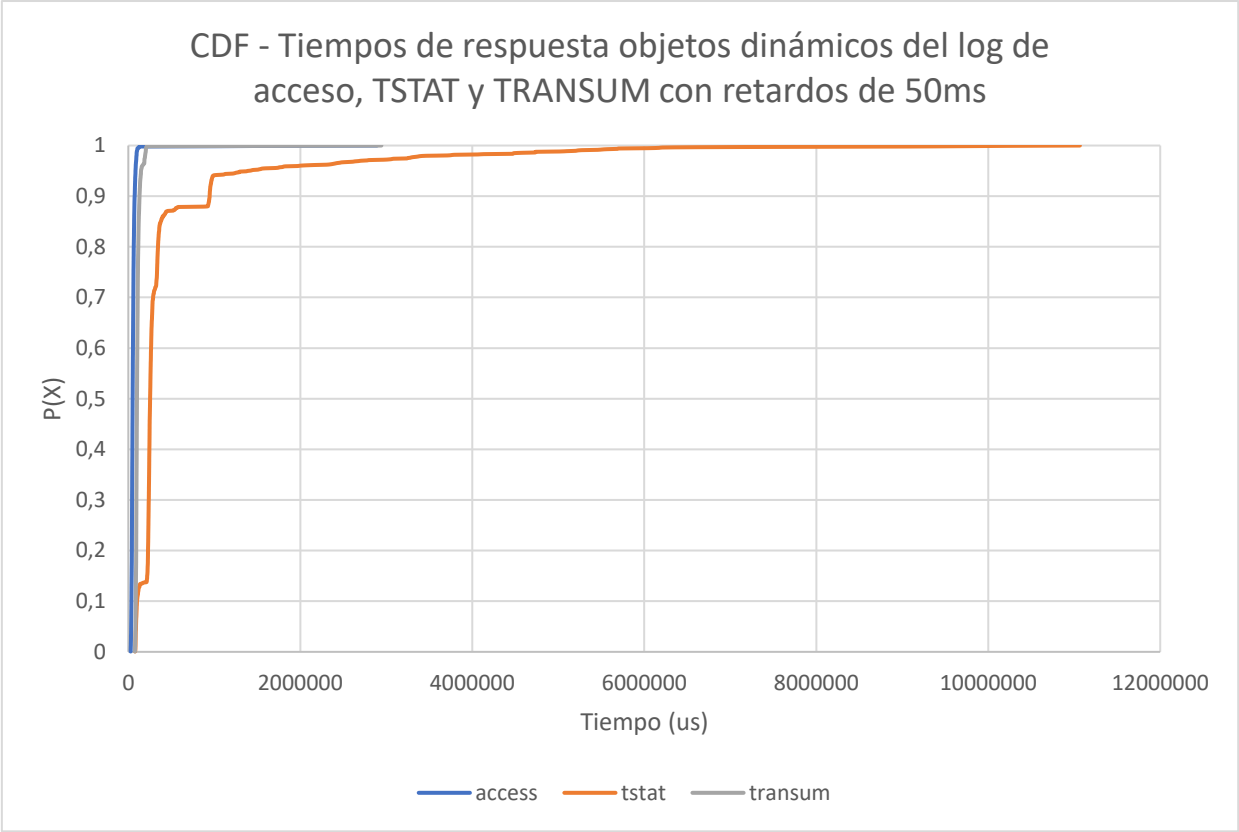

Figura 6-13: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con retardos de 50ms) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

Observando estas tres gráficas, podemos observar como ya al meter problemas en la red las distribuciones cambian drásticamente. En la Figura 6-11 observamos que son distribuciones parecidas, pero las obtenidas con las herramientas TRANSUM y TSTAT están desplazadas los 50 ms que se producen por los retardos. Esto mismo ocurre para la Figura 6-12.

Por último, podemos observar que en la Figura 6-13 la herramienta TSTAT obtiene tiempos demasiado elevados comparado con lo obtenido con TRANSUM. Esta última vuelve a obtener tiempos más parecidos a los del *log* de acceso.

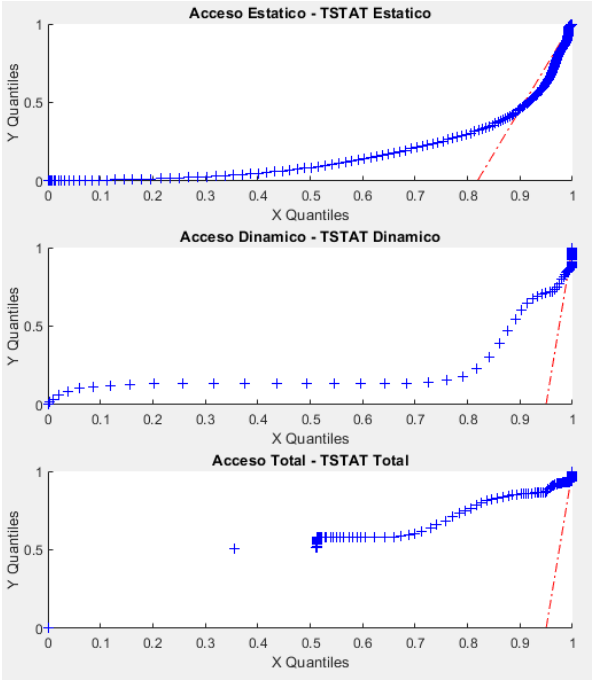

Figura 6-14: qqplots sobre el *log* de acceso del servidor HTTP frente al *log* HTTP de TSTAT (con retardos de 50 ms)

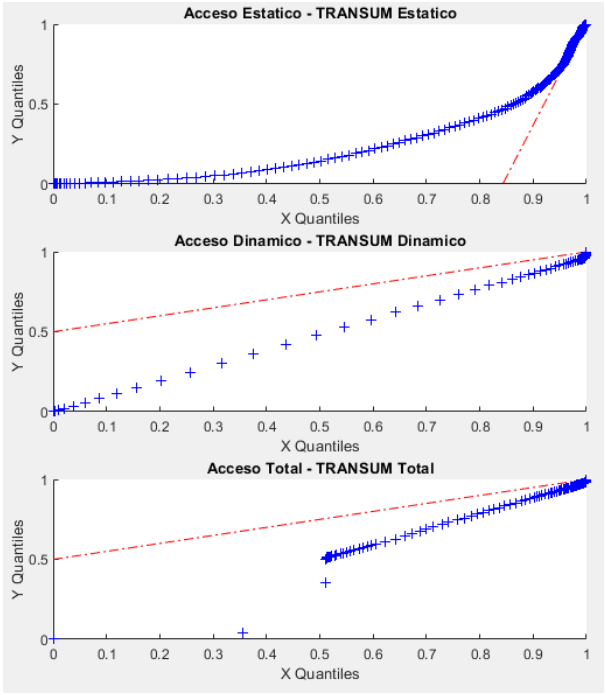

Figura 6-15: qqplots sobre el *log* de acceso del servidor HTTP frente al TRANSUM de Wireshark (con retardos de 50 ms)

Observando estas dos gráficas, podemos ver que ninguna de las dos herramientas sigue exactamente la distribución de los tiempos del *log* de acceso, pero la distribución de la herramienta TRANSUM es la más parecida.

Tabla 6-5: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el *log* HTTP de TSTAT (con retardos de 50 ms)

Métodos estadísticos	*Log* de acceso total vs TSTAT total	*Log* de acceso estático vs TSTAT estático	*Log* de acceso dinámico vs TSTAT dinámico
Test KS	1	1	1
Divergencia KL	1,6496e-4	0,0188	0,0043

Tabla 6-6: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el TRANSUM de Wireshark (con retardos de 50 ms)

Métodos estadísticos	*Log* de acceso total vs TRANSUM total	*Log* de acceso estático vs TRANSUM estático	*Log* de acceso dinámico vs TRANSUM dinámico
Test KS	1	1	1
Divergencia KL	1,5932e-4	0,0103	2,1236e-5

Con estas medidas realizadas para cada herramienta, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado un rechazo de la hipótesis nula en todos los casos, esto quiere decir que las distribuciones no son la misma. Y respecto a la divergencia KL podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con pérdidas de paquetes

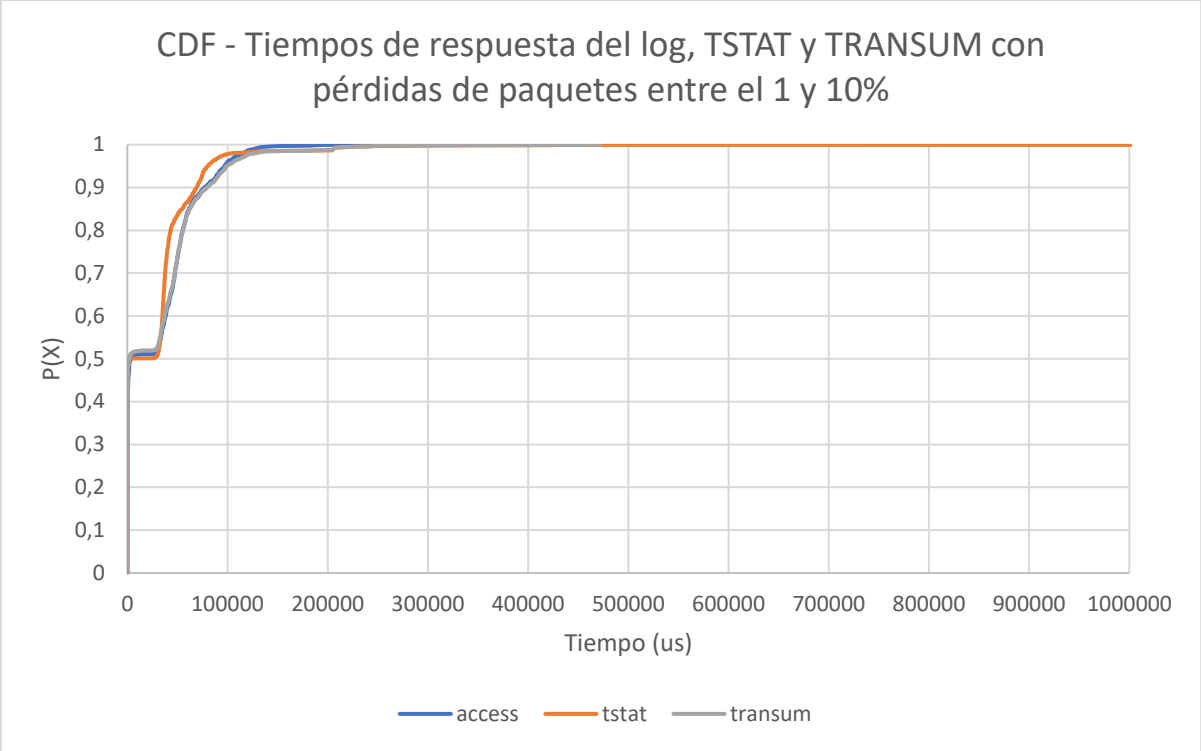

Figura 6-16: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con pérdidas entre el 1 y 10%) del *log* de acceso, TSTAT y TRANSUM

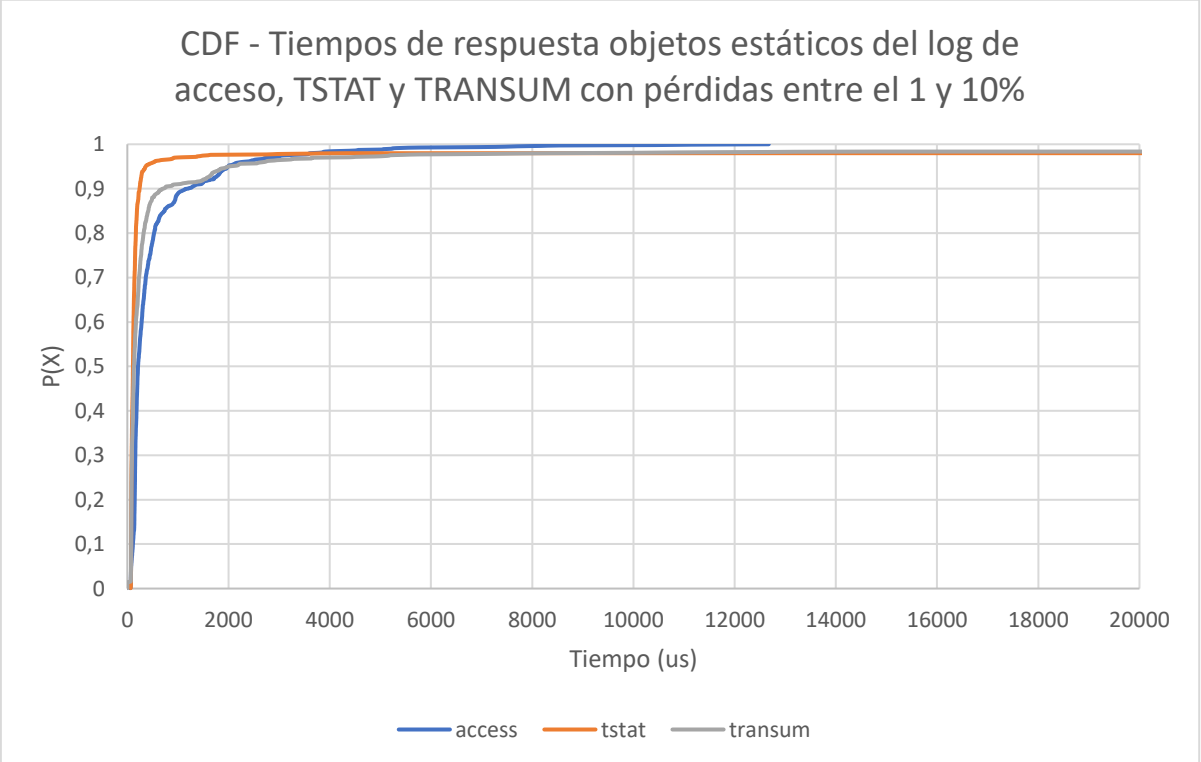

Figura 6-17: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con pérdidas entre el 1 y 10%) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

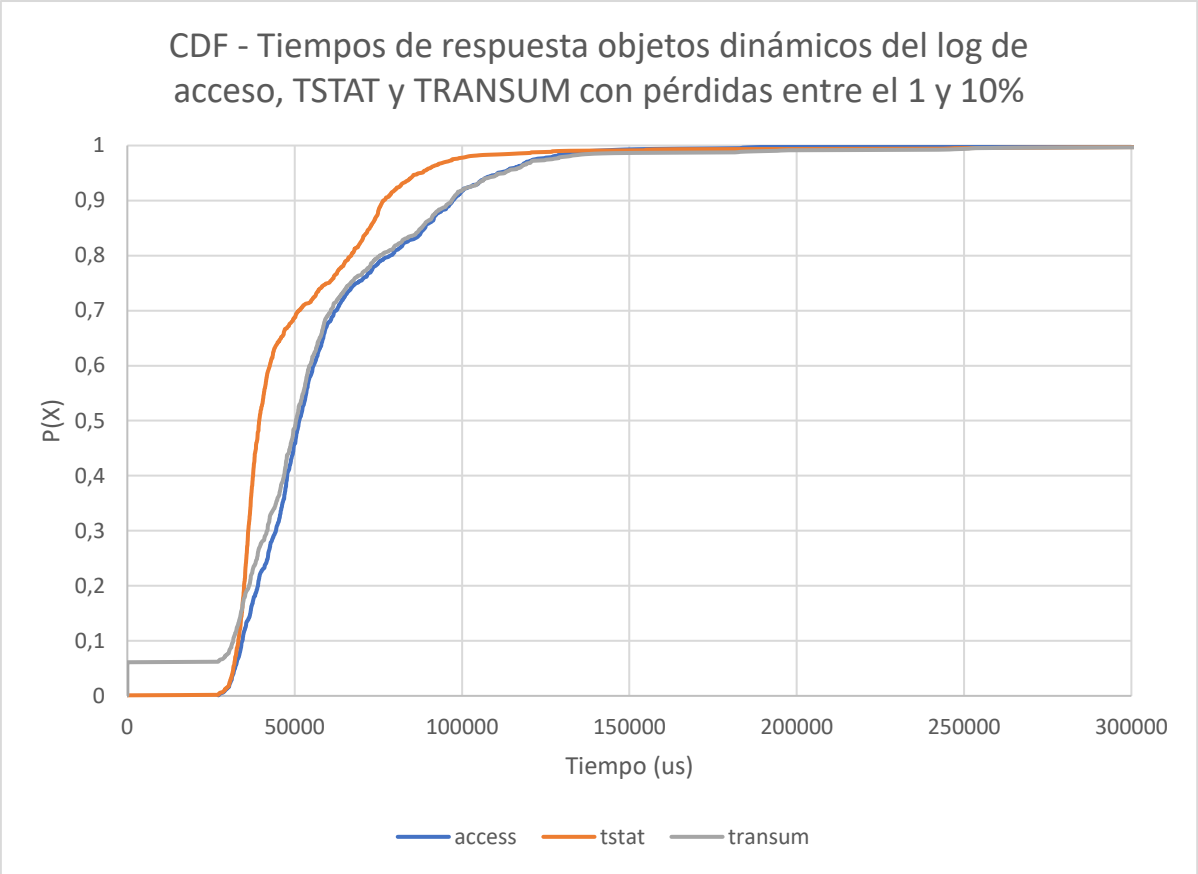

Figura 6-18: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con pérdidas entre el 1 y 10%) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

Si observamos las gráficas obtenidas con esta base de datos, podemos observar en la Figura 6-18 cómo la herramienta TRANSUM empieza un poco más arriba debido a las pérdidas de paquetes, pero se va asemejando a la distribución del *log* de acceso.

Por último, se puede observar de nuevo que la herramienta TRANSUM, en general, se asemeja más a la distribución del *log* de acceso.

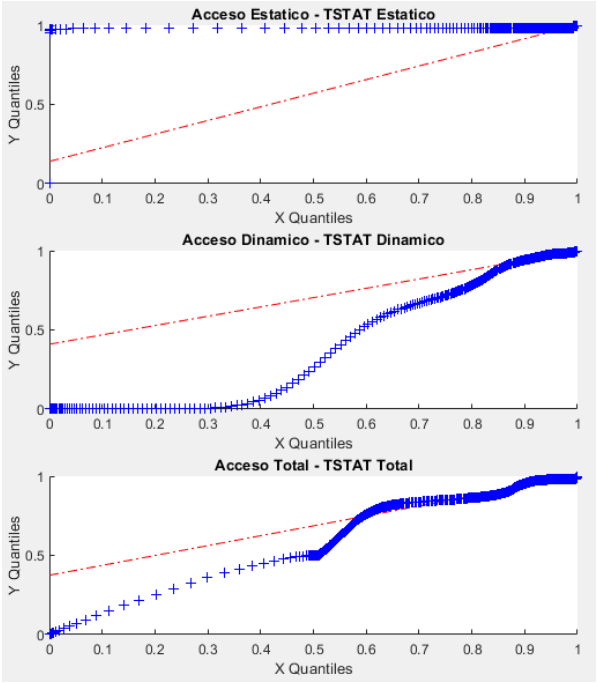

Figura 6-19: qqplots sobre el *log* de acceso del servidor HTTP frente al *log* HTTP de TSTAT (con pérdidas de paquetes entre el 1 y 10%)

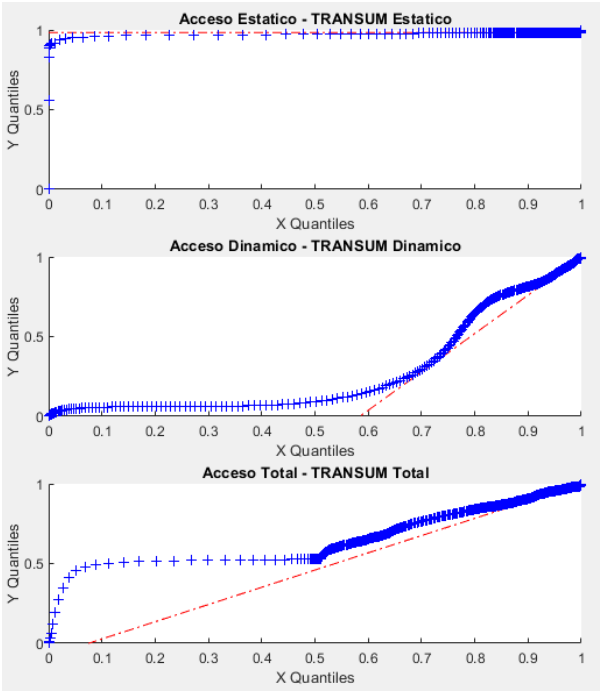

Figura 6-20:qqplots sobre el *log* de acceso del servidor HTTP frente al TRANSUM de Wireshark (con pérdidas de paquetes entre el 1 y 10%)

Observando estas gráficas, podemos observar que, en general, ninguna de las distribuciones de las herramientas TSTAT y TRANSUM sigue la distribución del *log* de acceso, pero la del TRANSUM sigue siendo la que más se asemeja.

Tabla 6-7: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el *log* HTTP de TSTAT (con pérdidas de paquetes entre el 1 y 10%)

Métodos estadísticos	*Log* de acceso total vs TSTAT total	*Log* de acceso estático vs TSTAT estático	*Log* de acceso dinámico vs TSTAT dinámico
Test KS	1	1	1
Divergencia KL	7,7933e-4	0,0149	0,0319

Tabla 6-8: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el TRANSUM de Wireshark (con pérdidas de paquetes entre el 1 y 10%)

Métodos estadísticos	*Log* de acceso total vs TRANSUM total	*Log* de acceso estático vs TRANSUM estático	*Log* de acceso dinámico vs TRANSUM dinámico
Test KS	1	1	1
Divergencia KL	0,0017	0,0141	0,0222

Con estas medidas realizadas para cada herramienta, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas y en la base de datos anterior, es decir, la prueba KS da como resultado un rechazo de la hipótesis nula en todos los casos, indicando que ninguna distribución se asemeja. Y respecto a la divergencia KL podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con retardos y limitación en ancho de banda

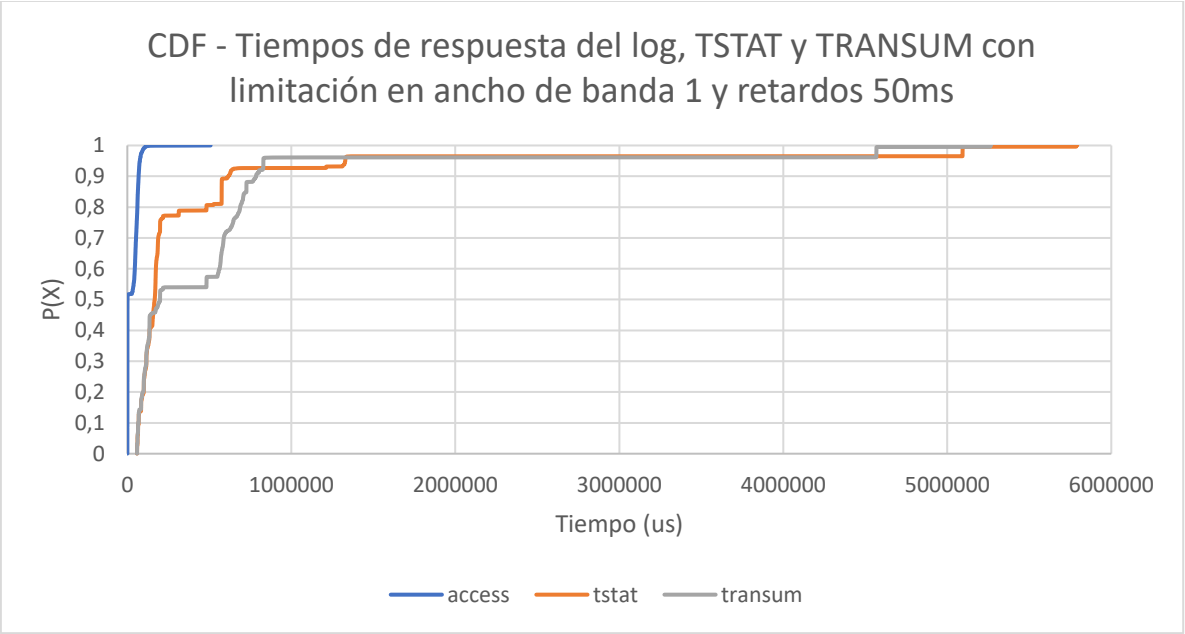

Figura 6-21: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con limitación en ancho de banda 1 Mbit/s y retardos 50ms) del *log* de acceso, TSTAT y TRANSUM

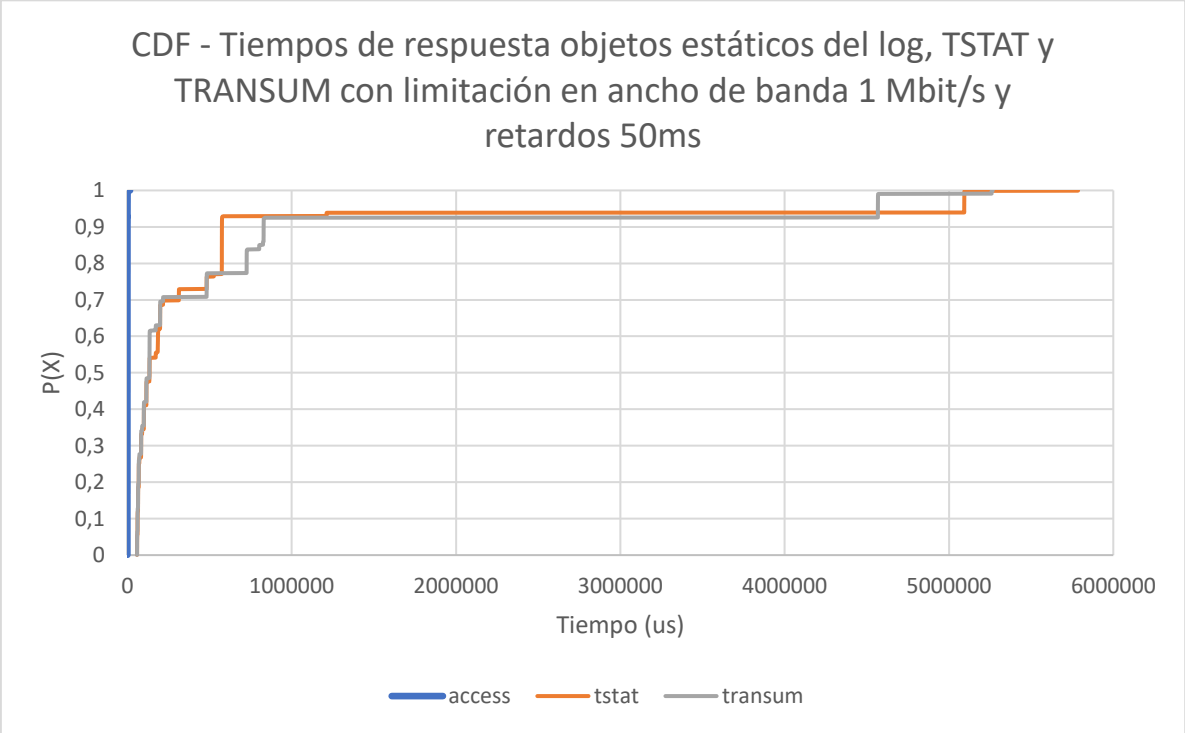

Figura 6-22: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con limitación en ancho de banda 1 Mbit/s y retardos 50ms) de objetos estáticos del *log* de acceso, TSTAT y TRANSUM

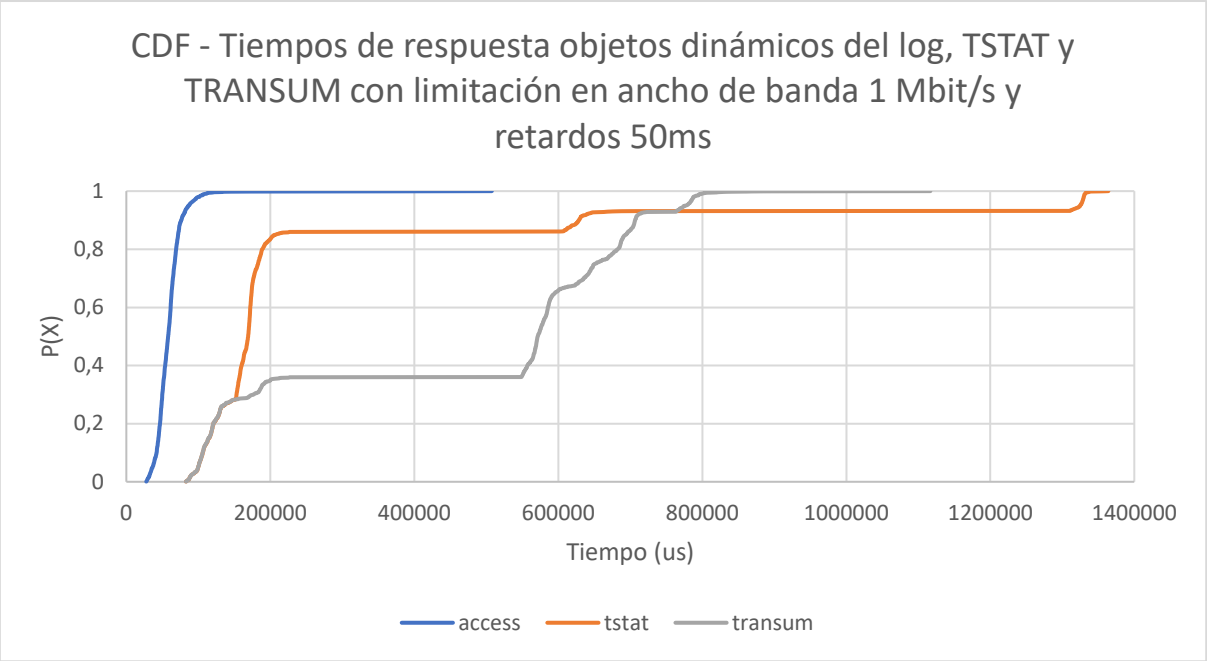

Figura 6-23: CDF de los tiempos de respuesta del servidor HTTP entre peticiones (con limitación en ancho de banda 1 Mbit/s y retardos 50ms) de objetos dinámicos del *log* de acceso, TSTAT y TRANSUM

Para esta base de datos, podemos observar cómo de diferentes son las tres distribuciones debido a los retardos, las distribuciones de las herramientas TSTAT y TRANSUM aparecen desplazadas dicho tiempo de retardo, mientras que en el resto de la gráfica podemos observar la diferencia de tiempos debido a la limitación del ancho de banda. Como en estos casos se producen retransmisiones, el tiempo de respuesta percibido por el cliente es mucho mayor. Cabe destacar que, mientras que el cliente está observando que está teniendo problemas a la hora de visualizar la página web, el servidor es ajeno a esta situación y no es consciente de que el cliente está observando problemas al acceder al servidor.

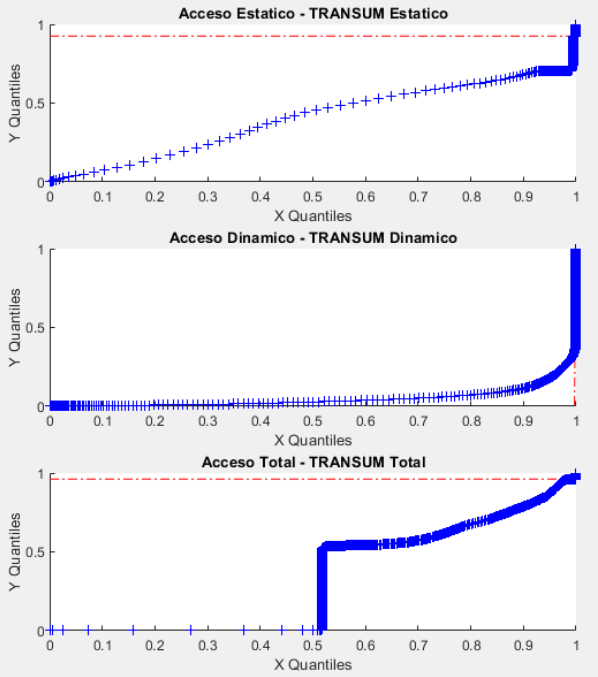

Figura 6-24: qqplots sobre el *log* de acceso del servidor HTTP frente al *log* HTTP de TSTAT (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

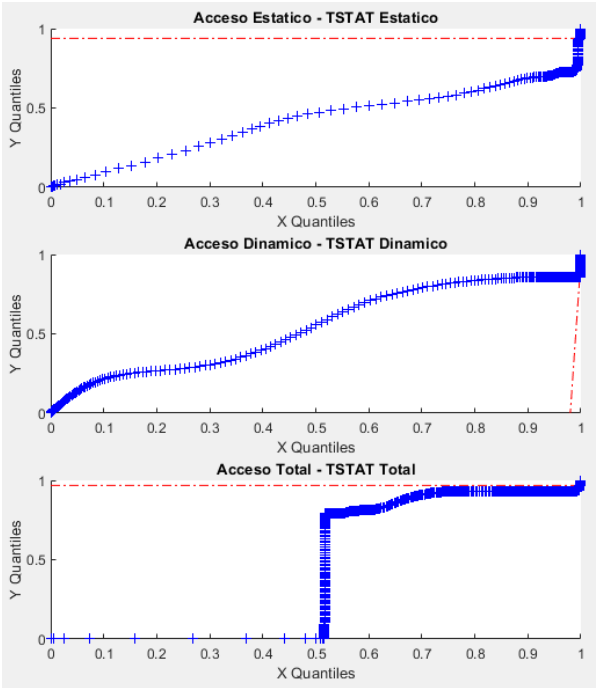

Figura 6-25: qqplots sobre el *log* de acceso del servidor HTTP frente al TRANSUM de Wireshark (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Con estas dos gráficas podemos observar lo comentado con las CDFs, ninguna de las distribuciones de las herramientas TRANSUM y TSTAT siguen la distribución del *log* de acceso.

Tabla 6-9: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el *log* HTTP de TSTAT (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Métodos estadísticos	*Log* de acceso total vs TSTAT total	*Log* de acceso estático vs TSTAT estático	*Log* de acceso dinámico vs TSTAT dinámico
Test KS	1	1	1
Divergencia KL	0,0188	0,0017	0,0027

Tabla 6-10: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTP y el TRANSUM de Wireshark (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Métodos estadísticos	*Log* de acceso total vs TRANSUM total	*Log* de acceso estático vs TRANSUM estático	*Log* de acceso dinámico vs TRANSUM dinámico
Test KS	1	1	1
Divergencia KL	0,0386	0,0026	0,1776

Por último, con estas medidas realizadas para cada herramienta, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas y sucede lo que ocurría en las dos pasadas bases de datos, son tan diferentes las distribuciones que la prueba KS rechaza la hipótesis nula debido a las diferencias tan grandes y, en cuanto a la divergencia KL, estos números cuanto mayores son menos se parecen las distribuciones entre sí.

6.3. Evaluación del Servidor HTTPS

Las pruebas realizadas con el servidor HTTPS han sido las de aplicar el algoritmo y sus adaptaciones a las bases de datos generadas. A continuación, mostraremos los resultados sobre todas las bases de datos:

- Resultados sobre la base de datos: Tráfico Ideal

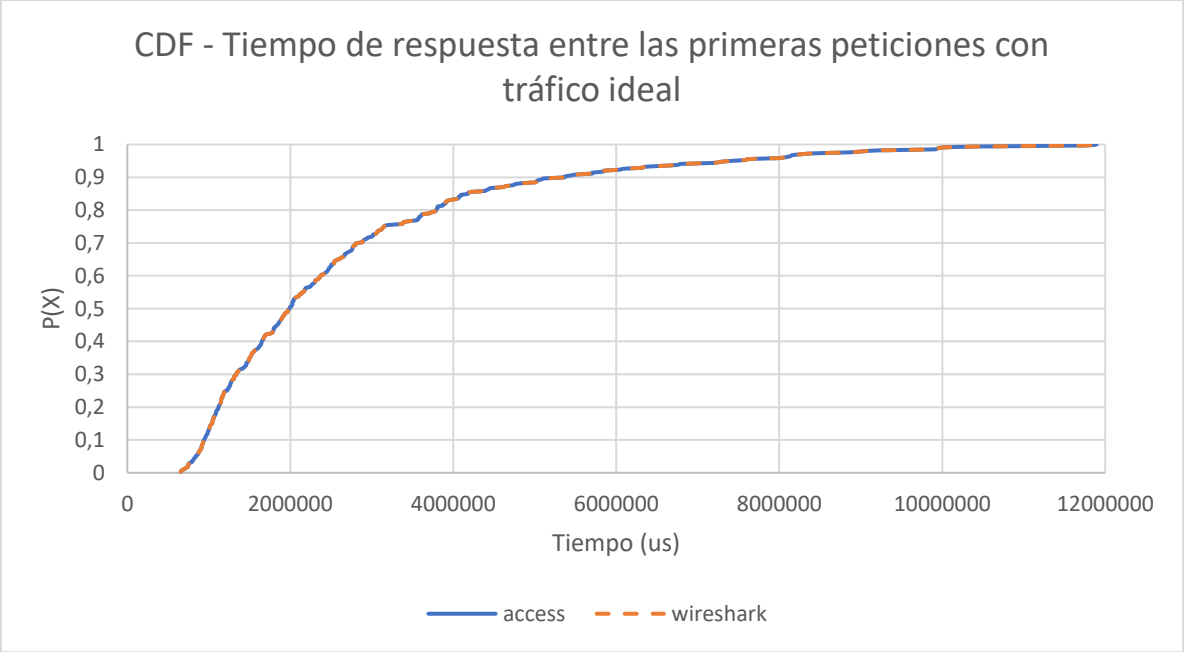

Figura 6-26: CDF de los tiempos de respuesta del servidor HTTPS entre las primeras peticiones del *log* de acceso y la captura realizada (con tráfico ideal)

Figura 6-27: CDF de los tiempos de respuesta del servidor HTTPS entre la primera petición y su última respuesta del *log* de acceso y la captura realizada (con tráfico ideal)

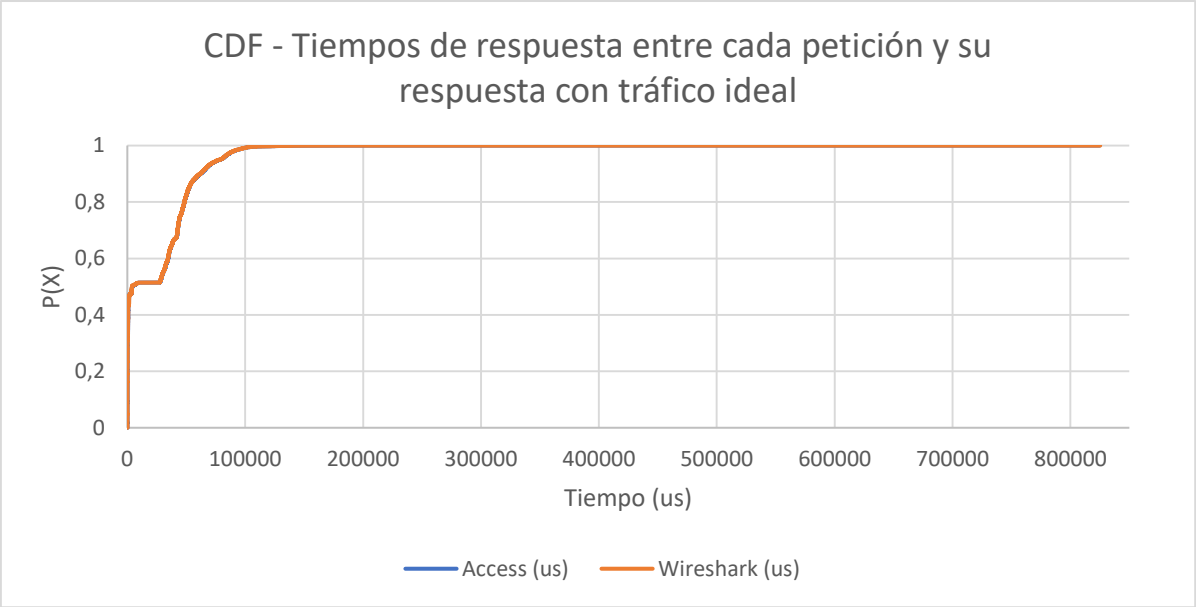

Figura 6-28: CDF de los tiempos de respuesta del servidor HTTPS entre cada petición y su respuesta del *log* de acceso y la captura realizada (con tráfico ideal)

Como se puede observar en estas tres gráficas, la distribución de los tiempos obtenidos con la herramienta Wireshark coincide con la distribución de los tiempos del *log* de acceso del servidor.

En la Figura 6-29 podemos ver cómo prácticamente son idénticas ambas distribuciones.

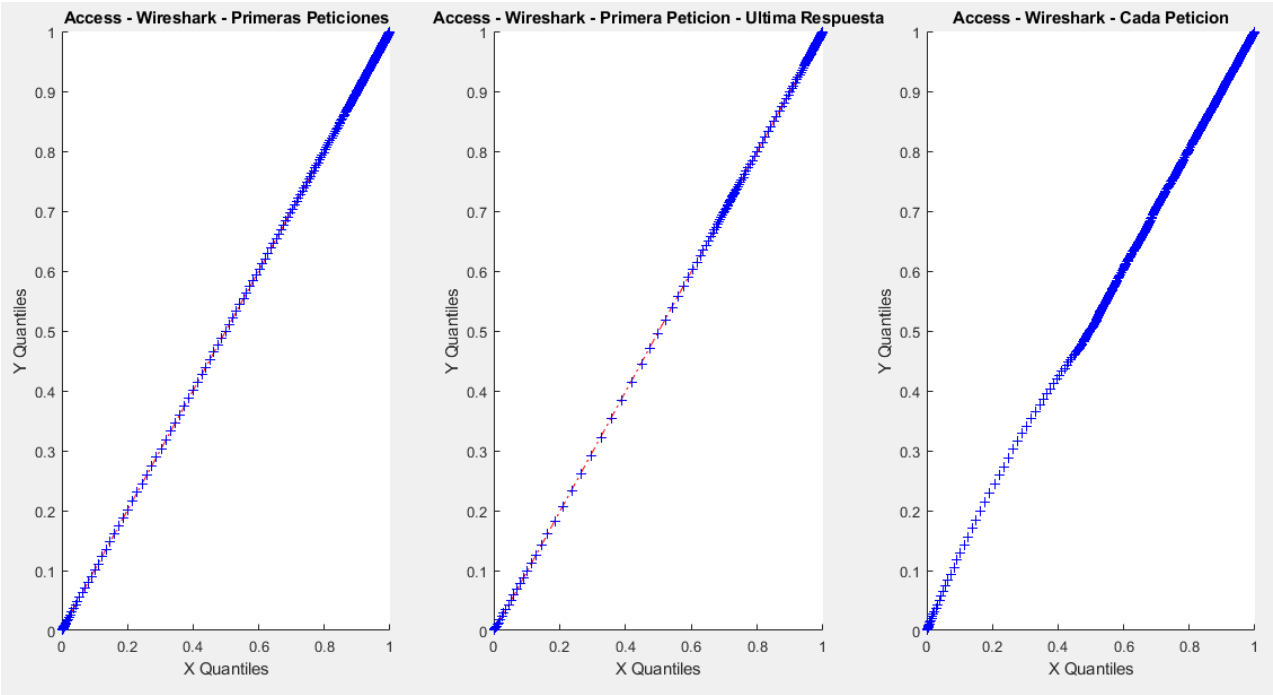

Figura 6-29: qqplots sobre el *log* de acceso del servidor HTTPS frente a la captura de Wireshark (con tráfico ideal)

Tabla 6-11: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTPS y la captura realizada (con tráfico ideal)

Métodos estadísticos	*Log* de acceso total vs Captura cada petición	*Log* de acceso estático vs Captura primeras peticiones	*Log* de acceso dinámico vs Captura primera petición – última respuesta
Test KS	0	0	0
Divergencia KL	9,2011e-6	9,2302e-8	3,6081e-6

Con estas medidas realizadas, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado una hipótesis nula en todos los casos, esto quiere decir que ambas distribuciones son prácticamente idénticas. Y respecto a la divergencia KL, podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con peticiones erróneas

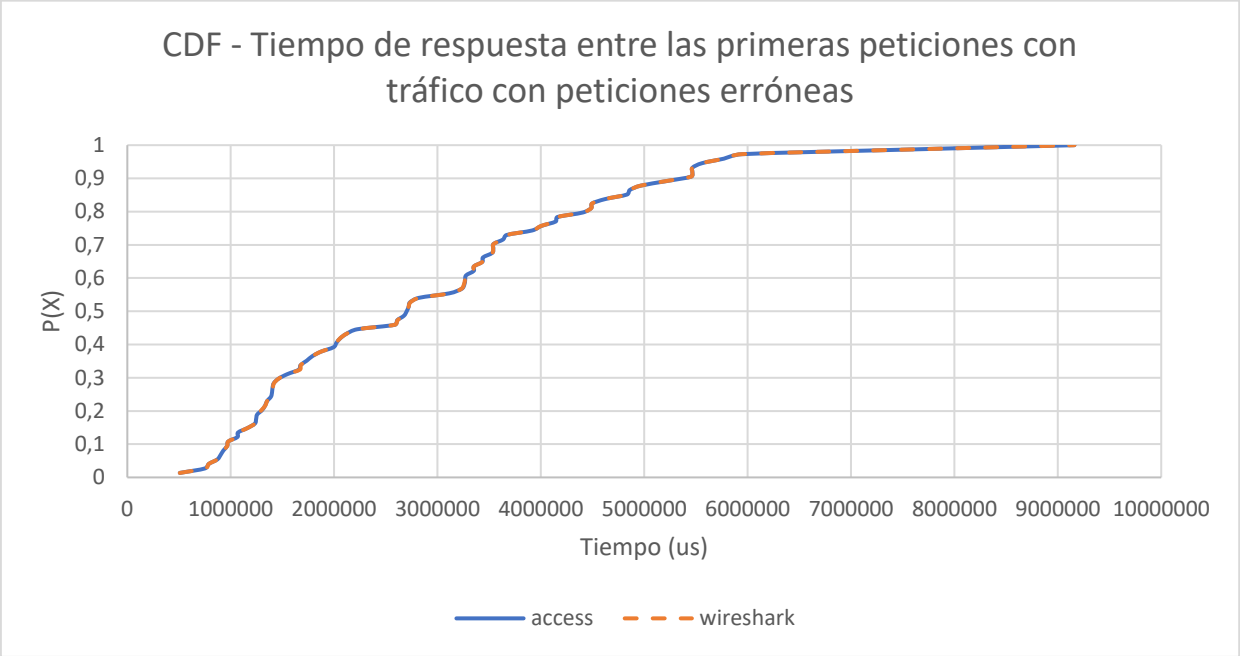

Figura 6-30: CDF de los tiempos de respuesta del servidor HTTPS entre las primeras peticiones del *log* de acceso y la captura realizada (con peticiones erróneas)

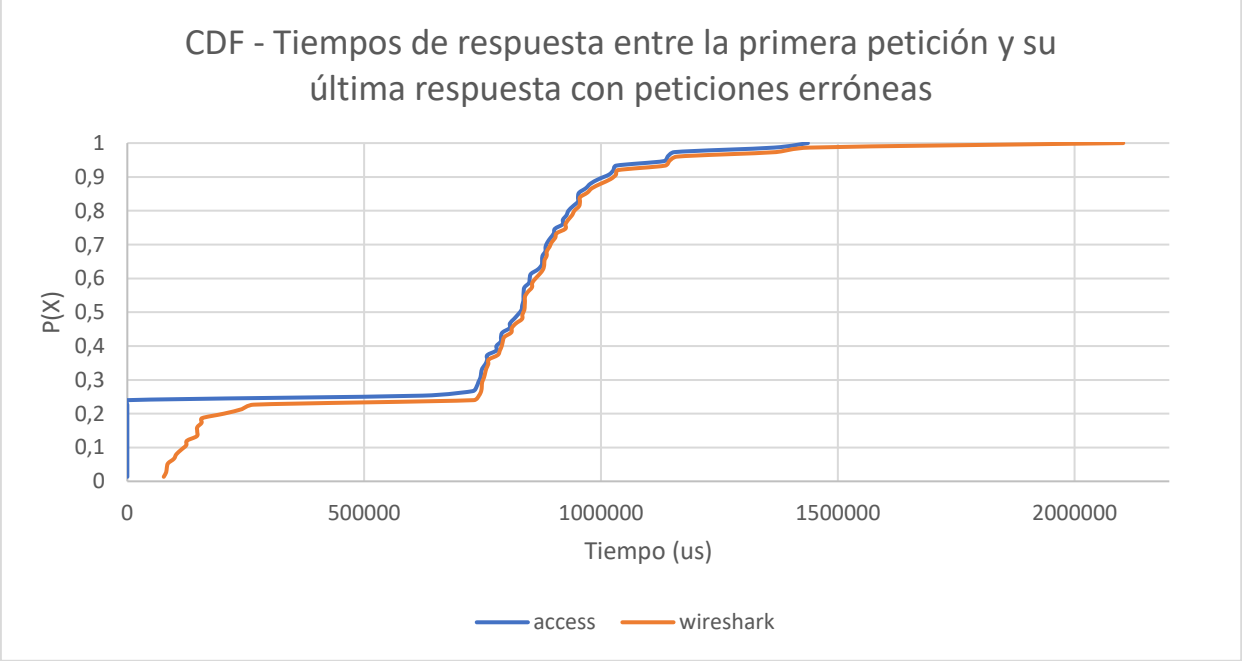

Figura 6-31: CDF de los tiempos de respuesta del servidor HTTPS entre la primera petición y su última respuesta del *log* de acceso y la captura realizada (con peticiones erróneas)

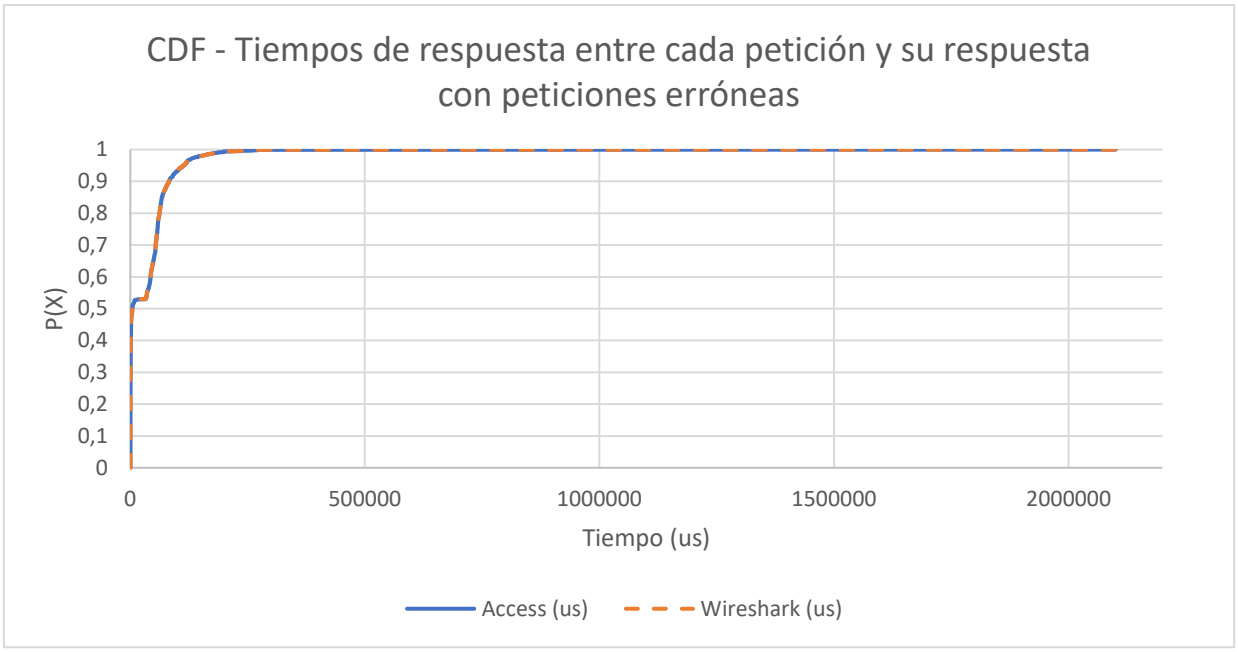

Figura 6-32: CDF de los tiempos de respuesta del servidor HTTPS entre cada petición y su respuesta del *log* de acceso y la captura realizada (con peticiones erróneas)

En estas tres gráficas podemos observar que todas las distribuciones se asemejan bastante, a excepción de la CDF de los tiempos de respuesta entre la primera petición y su última respuesta, en este caso podemos observar en la Figura 6-31 claramente la cantidad de peticiones erróneas que se han generado.

En la Figura 6-33 podemos observar cómo las distribuciones de los tiempos entre las primeras peticiones y entre cada petición sigue la distribución del *log* de acceso, mientras que los tiempos entre la primera petición y su última respuesta se observa claramente que no sigue con dicha distribución.

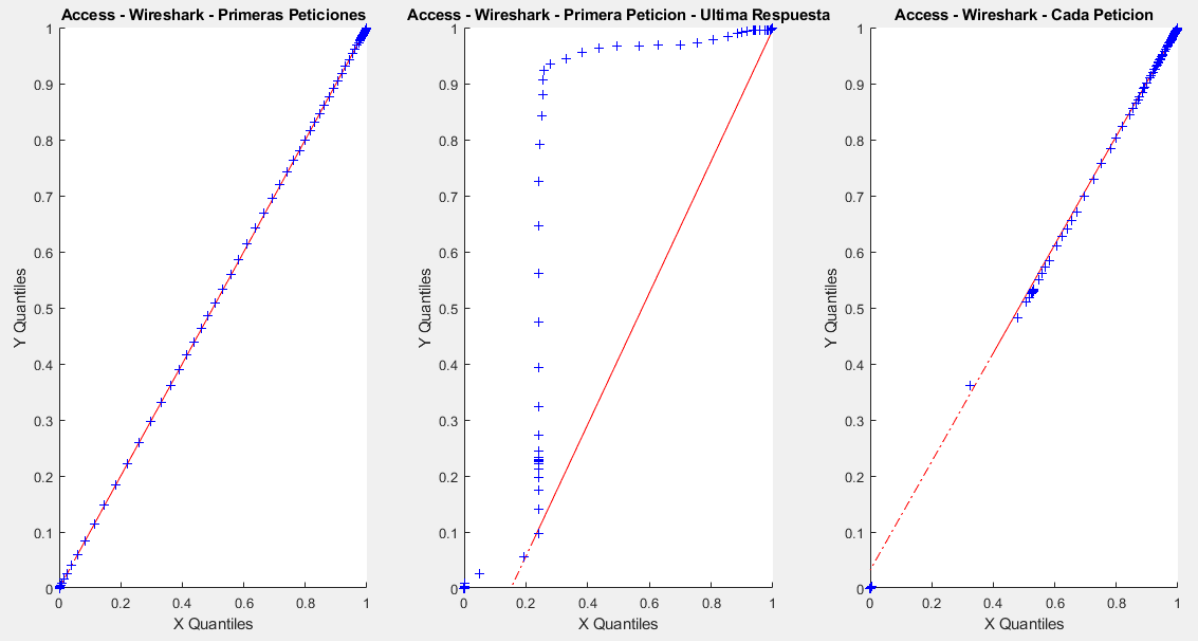

Figura 6-33: qqplots sobre el *log* de acceso del servidor HTTPS frente a la captura de Wireshark (con peticiones erróneas)

Tabla 6-12: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTPS y la captura realizada (con peticiones erróneas)

Métodos estadísticos	*Log* de acceso total vs Captura cada petición	*Log* de acceso estático vs Captura primeras peticiones	*Log* de acceso dinámico vs Captura primera petición – última respuesta
Test KS	0	0	1
Divergencia KL	6,4914e-6	7,2659e-8	0,0577

Con estas medidas realizadas, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado una hipótesis nula en los casos de los tiempos entre cada petición y entre las primeras peticiones, mientras que no rechaza la hipótesis nula para el caso de la primera petición y su última respuesta. Respecto a la divergencia KL, podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con retardos

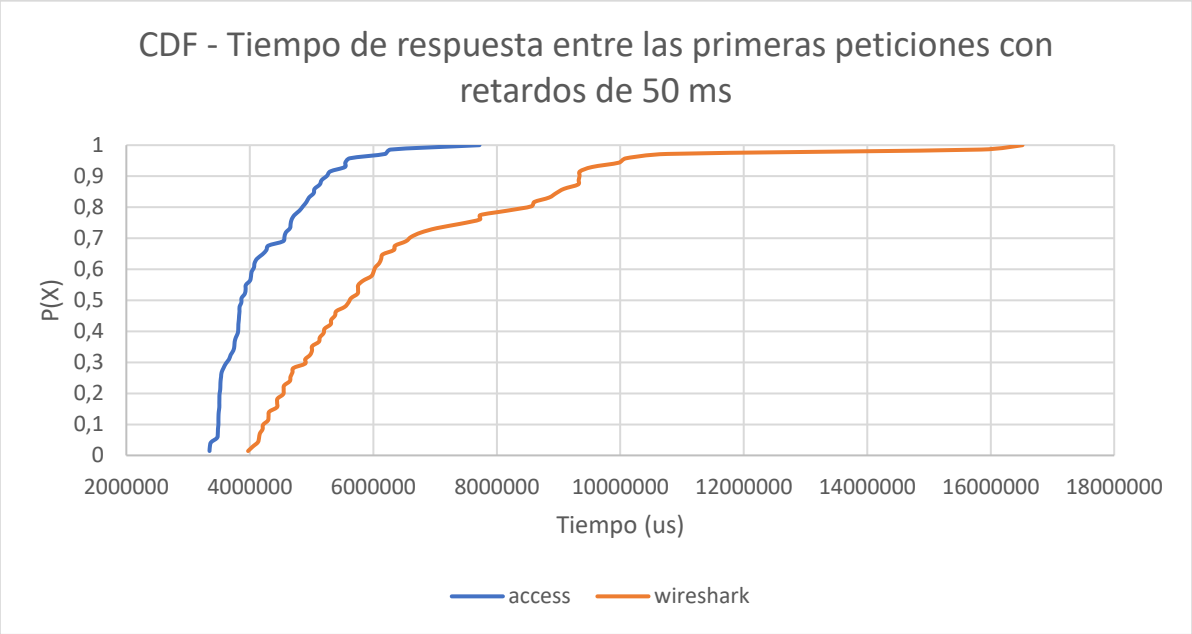

Figura 6-34: CDF de los tiempos de respuesta del servidor HTTPS entre las primeras peticiones del *log* de acceso y la captura realizada (con retardos de 50 ms)

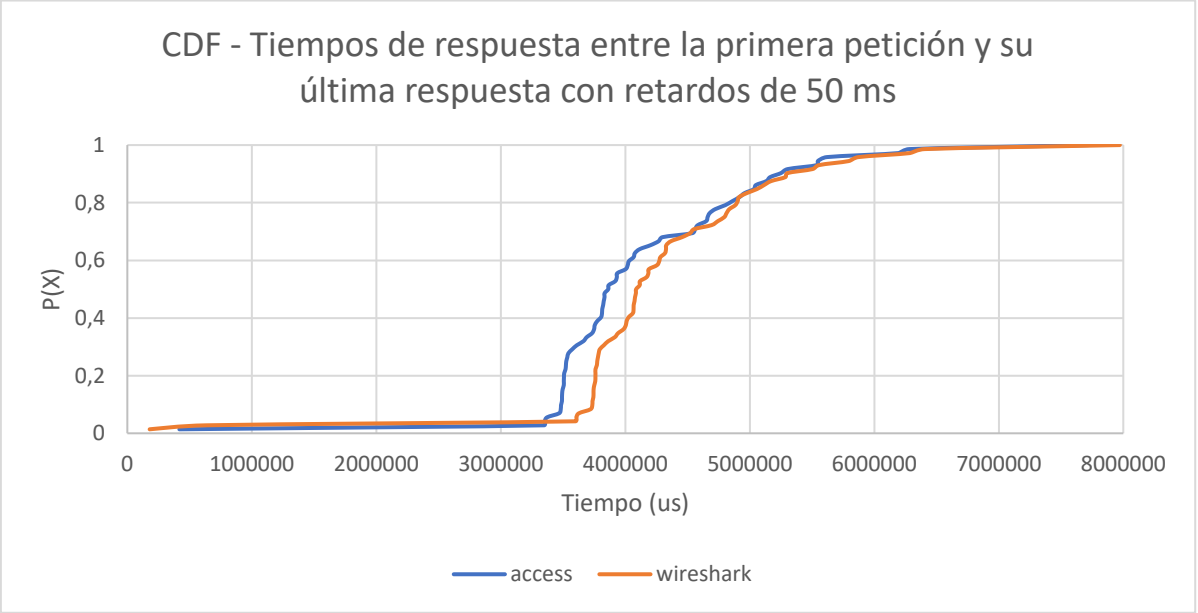

Figura 6-35: CDF de los tiempos de respuesta del servidor HTTPS entre la primera petición y su última respuesta del *log* de acceso y la captura realizada (con retardos de 50 ms)

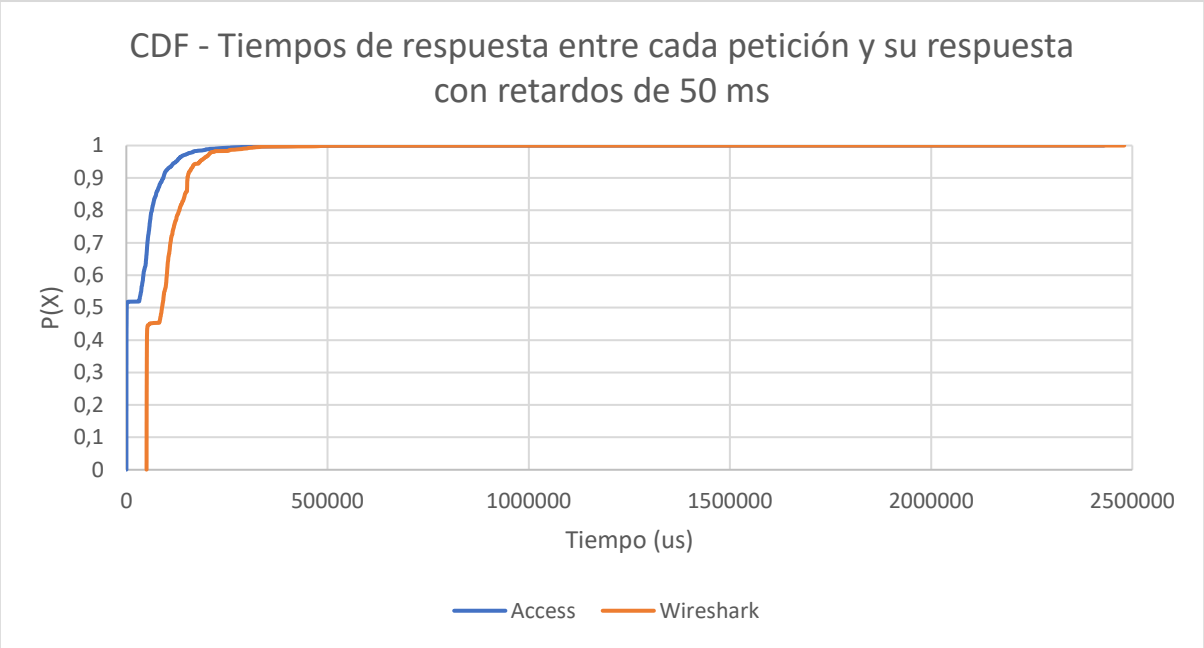

Figura 6-36: CDF de los tiempos de respuesta del servidor HTTPS entre cada petición y su respuesta del *log* de acceso y la captura realizada (con retardos de 50 ms)

En esta base de datos ocurre una situación parecida a la ocurrida con esta misma base de datos en el servidor HTTP, podemos observar que la distribución de los tiempos de la herramienta Wireshark aparece desplazada el tiempo de los retardos, pero se asemeja bastante a la del *log* de acceso.

En la Figura 6-37 podemos observar que la distribución de los tiempos de las primeras peticiones es la que más se parece a la distribución de los tiempos del *log* de acceso.

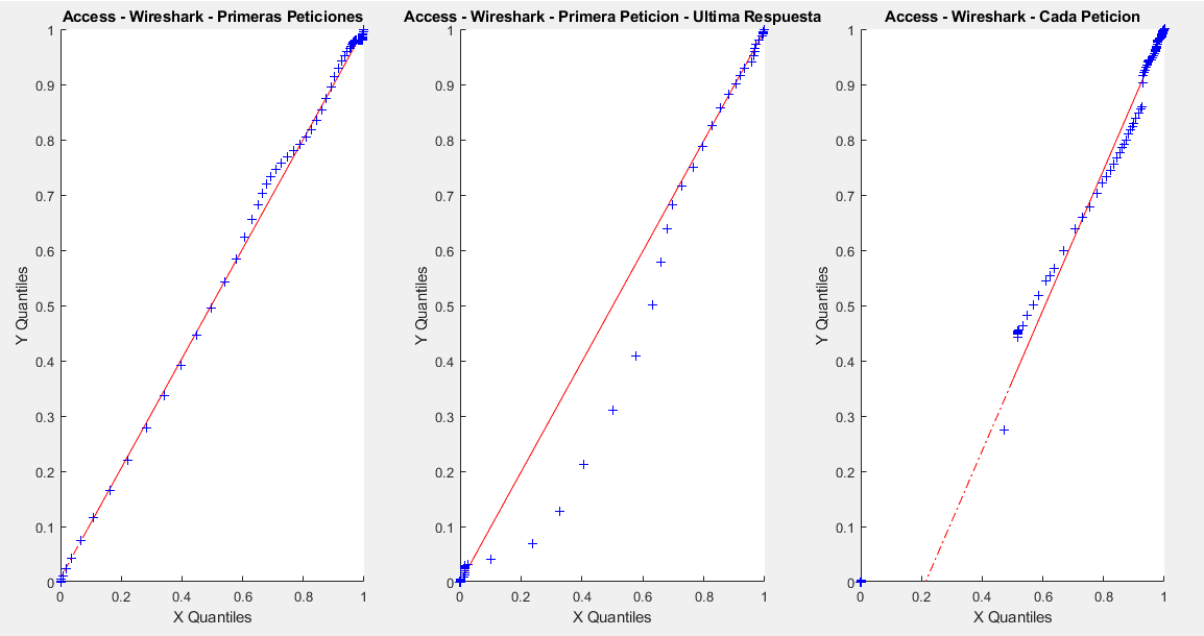

Figura 6-37: qqplots sobre el *log* de acceso del servidor HTTPS frente a la captura de Wireshark (con retardos de 50 ms)

Tabla 6-13: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTPS y la captura realizada (con retardos de 50 ms)

Métodos estadísticos	*Log* de acceso total vs Captura cada petición	*Log* de acceso estático vs Captura primeras peticiones	*Log* de acceso dinámico vs Captura primera petición – última respuesta
Test KS	1	0	1
Divergencia KL	0,0203	1,7046e-4	0,0119

Con estas medidas realizadas, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado una hipótesis nula en el caso de las primeras peticiones, mientras que no rechaza la hipótesis nula para los otros dos. Respecto a la divergencia KL, podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con pérdidas de paquetes

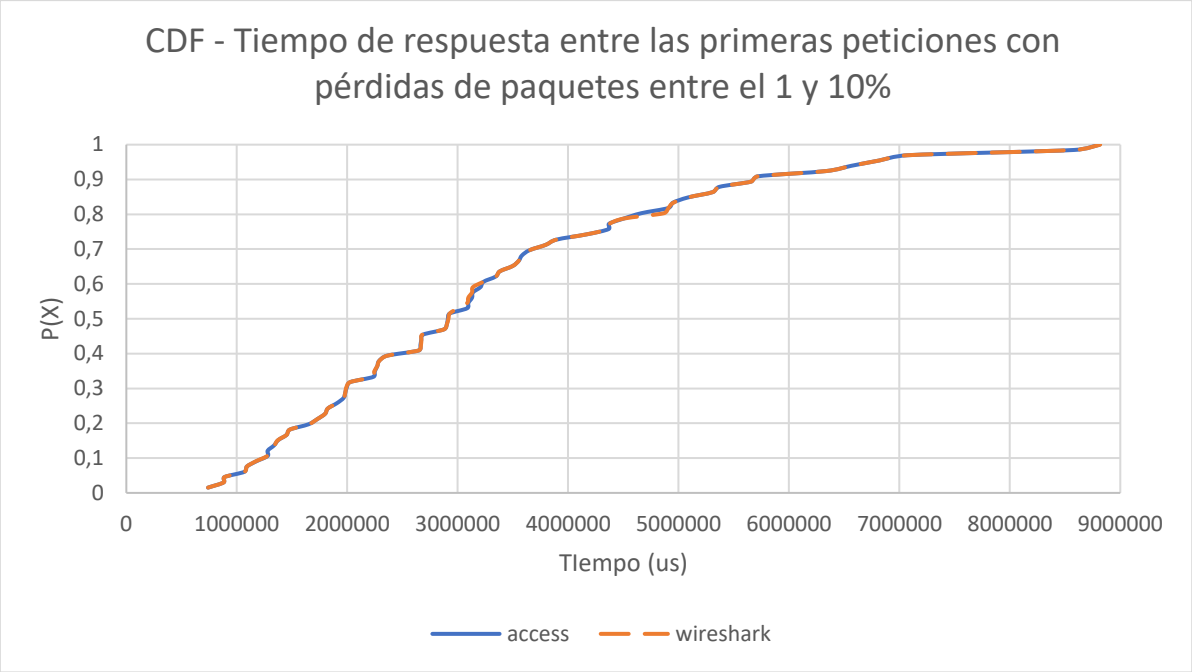

Figura 6-38: CDF de los tiempos de respuesta del servidor HTTPS entre las primeras peticiones del *log* de acceso y la captura realizada (con pérdidas de paquetes entre el 1 y 10%)

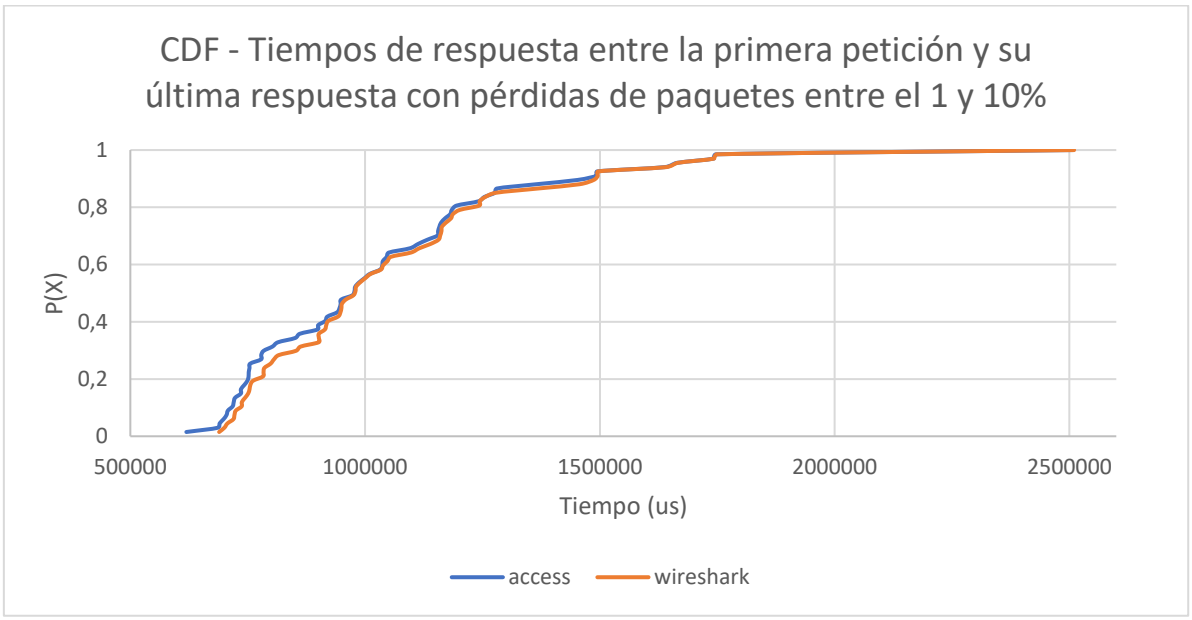

Figura 6-39: CDF de los tiempos de respuesta del servidor HTTPS entre la primera petición y su última respuesta del *log* de acceso y la captura realizada (con pérdidas de paquetes entre el 1 y 10%)

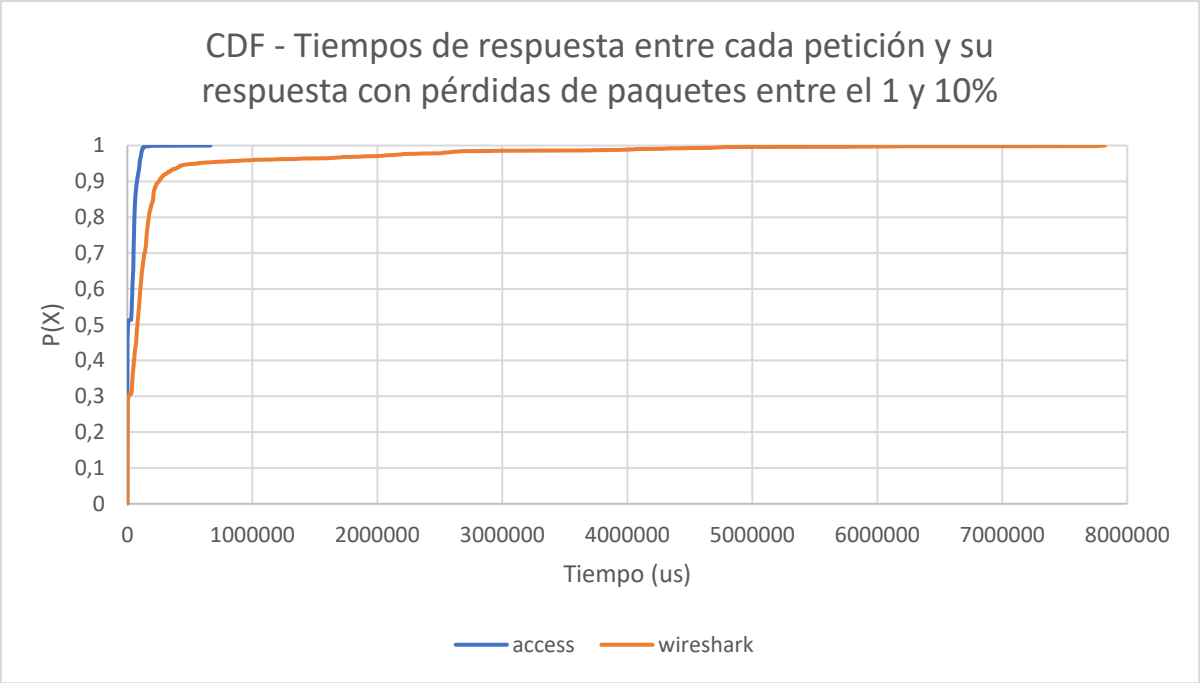

Figura 6-40: CDF de los tiempos de respuesta del servidor HTTPS entre cada petición y su respuesta del *log* de acceso y la captura realizada (con pérdidas de paquetes entre el 1 y 10%)

En esta última gráfica podemos observar cómo los tiempos obtenidos con la herramienta Wireshark son más altos que los registrados en el *log* de acceso, esto es debido a la pérdida de paquetes producida y las retransmisiones que se generan. En cuanto a las otras dos gráficas podemos observar que la distribución de tiempos de la herramienta Wireshark se asemeja más a la distribución de tiempos del *log* de acceso.

En la Figura 6-41 podemos observar lo comentado en las gráficas de las CDFs, la distribución que menos se parece a la del *log* de acceso es la distribución de los tiempos entre cada petición.

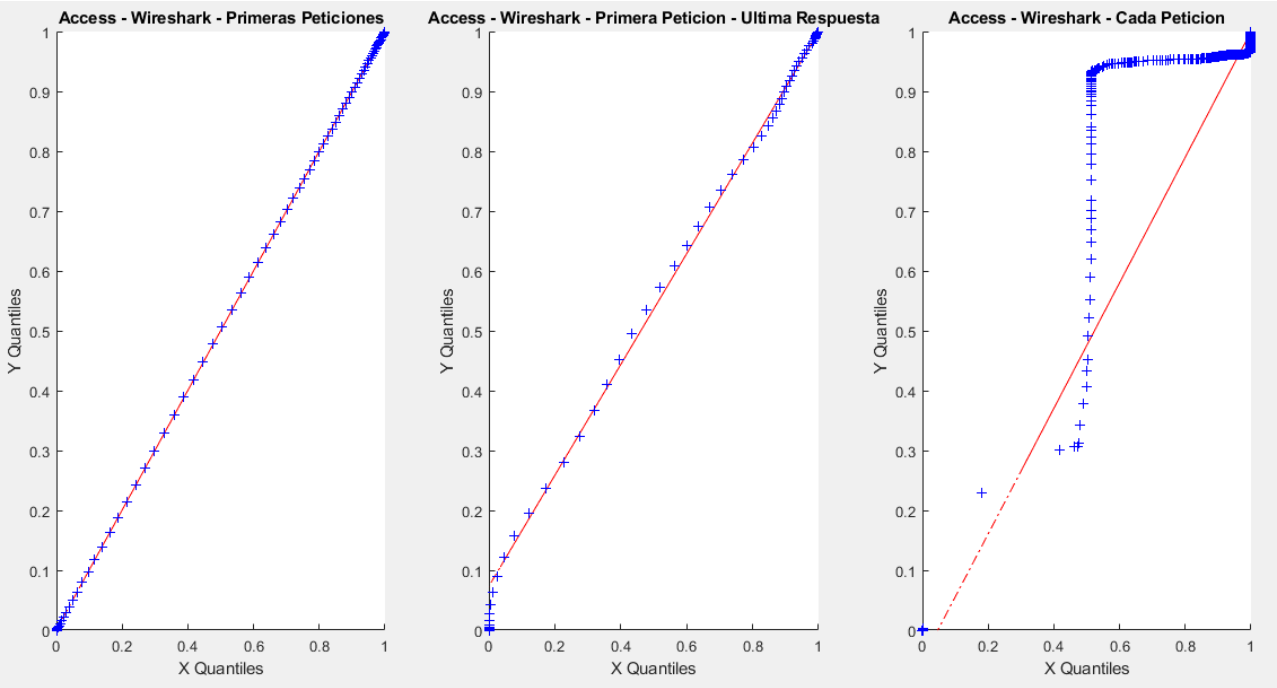

Figura 6-41: qqplots sobre el *log* de acceso del servidor HTTPS frente a la captura de Wireshark (con pérdidas de paquetes entre el 1 y 10%)

Tabla 6-14: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTPS y la captura realizada (con pérdidas de paquetes entre el 1 y 10%)

Métodos estadísticos	*Log* de acceso total vs Captura cada petición	*Log* de acceso estático vs Captura primeras peticiones	*Log* de acceso dinámico vs Captura primera petición – última respuesta
Test KS	1	0	0
Divergencia KL	0,006	4,974e-7	0,0012

Con estas medidas realizadas, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado un rechazo de la hipótesis nula en el caso de cada petición, mientras que rechaza la hipótesis nula para los otros dos. Respecto a la divergencia KL, podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

- Resultados sobre la base de datos: Tráfico con retardos y limitación en ancho de banda

Figura 6-42: CDF de los tiempos de respuesta del servidor HTTPS entre las primeras peticiones del *log* de acceso y la captura realizada (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

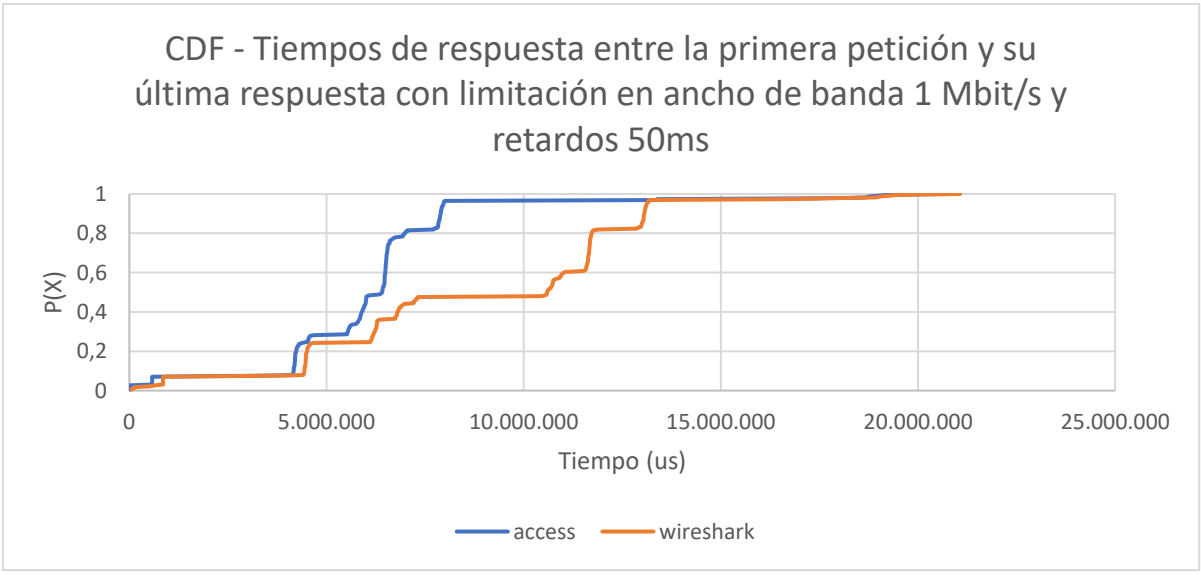

Figura 6-43: CDF de los tiempos de respuesta del servidor HTTPS entre la primera petición y su última respuesta del *log* de acceso y la captura realizada (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

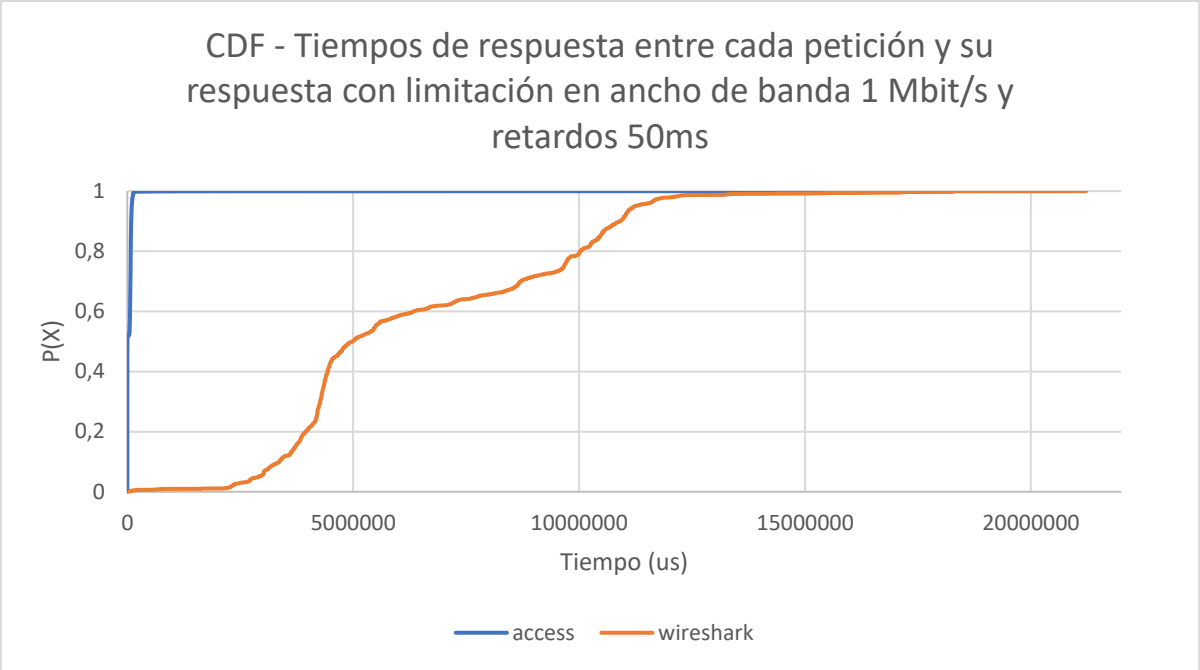

Figura 6-44: CDF de los tiempos de respuesta del servidor HTTPS entre cada petición y su respuesta del *log* de acceso y la captura realizada (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Con estas tres gráficas podemos observar que también sucede lo ocurrido con esta base de datos del servidor HTTP, el cliente observa una cosa y el servidor registra otra. En cuanto a la Figura 6-44 observamos que dichos tiempos calculados siguen la distribución de los tiempos del *log* de acceso, mientras que para las otras dos gráficas se observa claramente que no siguen dicha distribución.

En la Figura 6-45 podemos observar lo comentado anteriormente, la única distribución que se asemeja a la distribución del *log* de acceso es la de los tiempos entre las primeras peticiones.

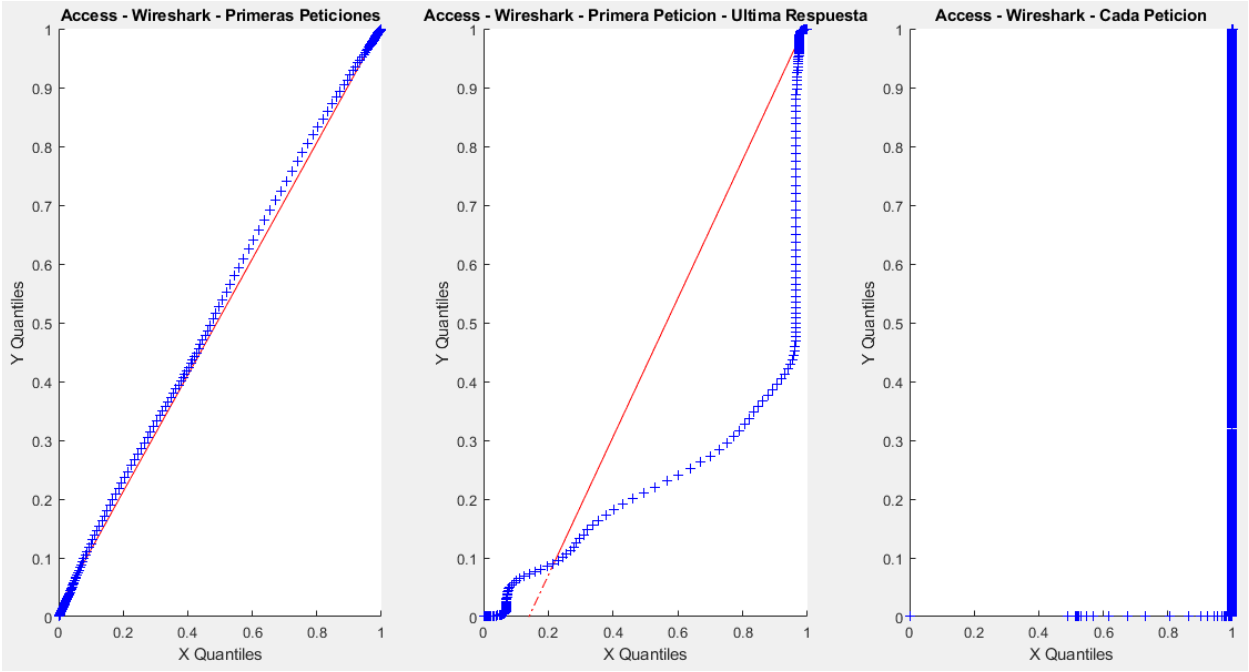

Figura 6-45: qqplots sobre el *log* de acceso del servidor HTTPS frente a la captura de Wireshark (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Tabla 6-15: Test KS y Divergencia KL entre el *log* de acceso del servidor HTTPS y la captura realizada (con limitación en ancho de banda 1 Mbit/s y retardos 50ms)

Métodos estadísticos	*Log* de acceso total vs Captura cada petición	*Log* de acceso estático vs Captura primeras peticiones	*Log* de acceso dinámico vs Captura primera petición – última respuesta
Test KS	1	0	1
Divergencia KL	0,8411	7,8743e-4	0,061

En cuanto a las bases de datos, con estas últimas medidas realizadas, podemos ver de una manera más detallada cuánto se parecen las distribuciones. Observamos que estos datos obtenidos coinciden con lo que hemos visto en las gráficas, es decir, la prueba KS da como resultado un rechazo de la hipótesis nula en el caso de cada petición, mientras que rechaza la hipótesis nula para los otros dos. Respecto a la divergencia KL, podemos observar que a menor número obtenido más se parecen las distribuciones gráficamente.

6.4. *Log* de acceso del servidor HTTP vs servidor HTTPS

En esta sección se va a mostrar la comparativa de los tiempos de respuesta que registra el *log* de acceso del servidor HTTP frente a los tiempos de respuesta que registra el *log* de acceso del servidor HTTPS.

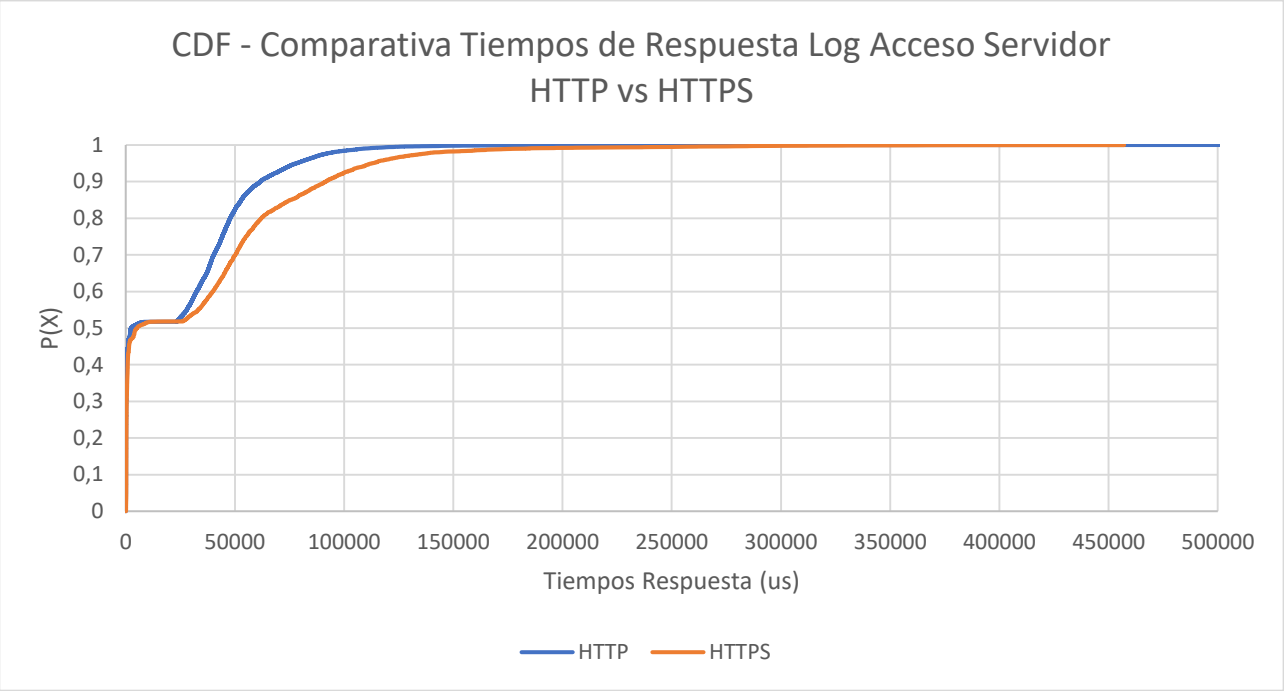

Figura 6-46: Comparativa de los tiempos de respuesta del *log* de acceso de servidor HTTP vs servidor HTTPS

Como podemos observar en la gráfica y viendo el comportamiento de las CDFs de los capítulos anteriores, en torno al 50% se producen peticiones estáticas y podemos ver que los tiempos son parecidos, aunque se observan mayores tiempos para el servidor HTTPS. En cuanto al porcentaje restante, se corresponden con peticiones dinámicas y observamos que los tiempos de respuesta del servidor HTTPS son mayores que los del servidor HTTP. Esta diferencia de tiempos es normal debido a que el servidor HTTPS necesita algo más de tiempo para procesar cada petición y respuesta al ser un protocolo cifrado. Aun así, aunque el tiempo de respuesta para este servidor sea algo mayor, es preferible que tu servidor trabaje sobre HTTPS que sobre HTTP para protegerse sobre posibles ataques de terceros.

6.5. Correlación de Registros

En esta sección se van a mostrar los datos que se trasladan a Elasticsearch y son mostrados gráficamente por Kibana. Con la estructura diseñada y comentada en la Figura 4-2, también podemos correlar y realizar la monitorización de ambos servidores.

A continuación, se muestran las siguientes gráficas para el servidor HTTP:´

- Resultados sobre la base de datos: Tráfico Ideal

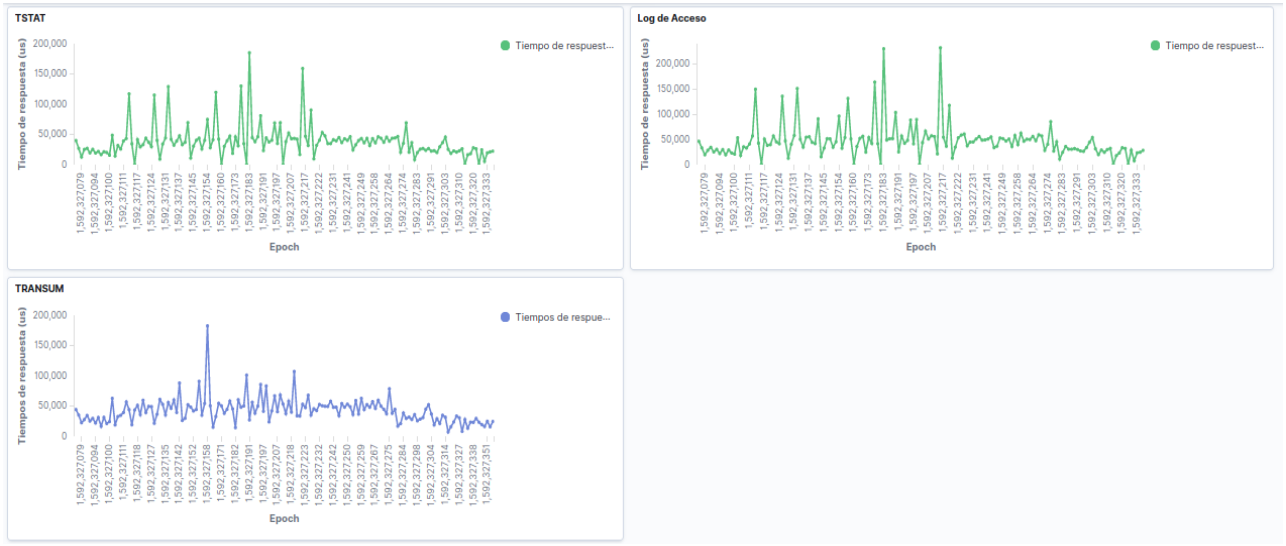

Figura 6-47: Dashboard de Kibana para monitorizar los tiempos de respuesta de la base de datos de tráfico ideal del servidor HTTP

- Resultados sobre la base de datos: Tráfico con retardos y limitación en ancho de banda

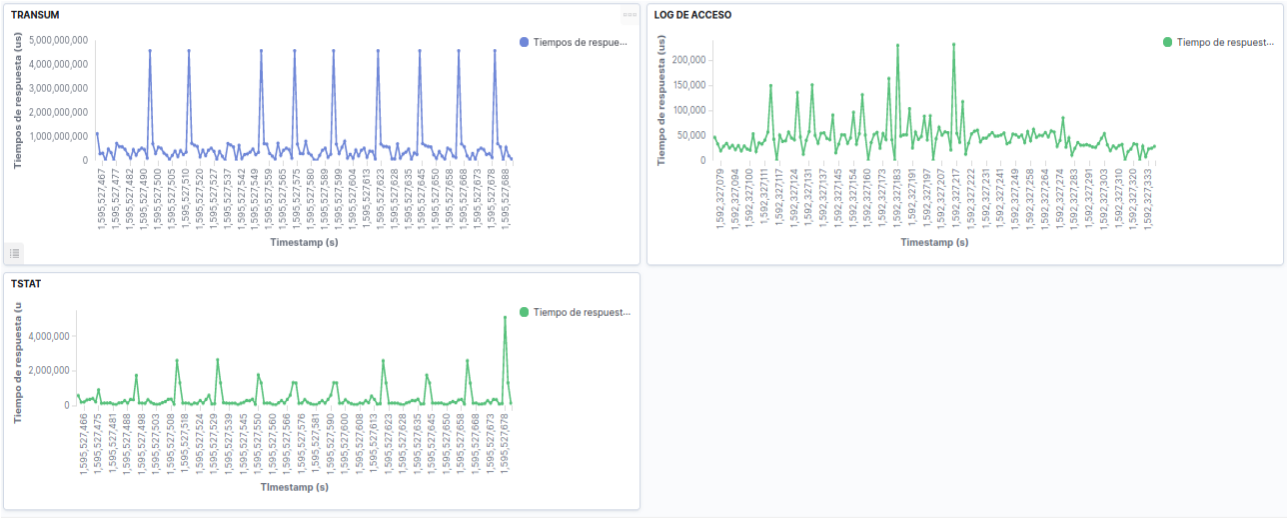

Figura 6-48: Dashboard de Kibana para monitorizar los tiempos de respuesta de la base de datos de tráfico con retardos y limitación en ancho de banda del servidor HTTP

Y, por último, se muestran las siguientes gráficas para el servidor HTTPS:

- Resultados sobre la base de datos: Tráfico Ideal

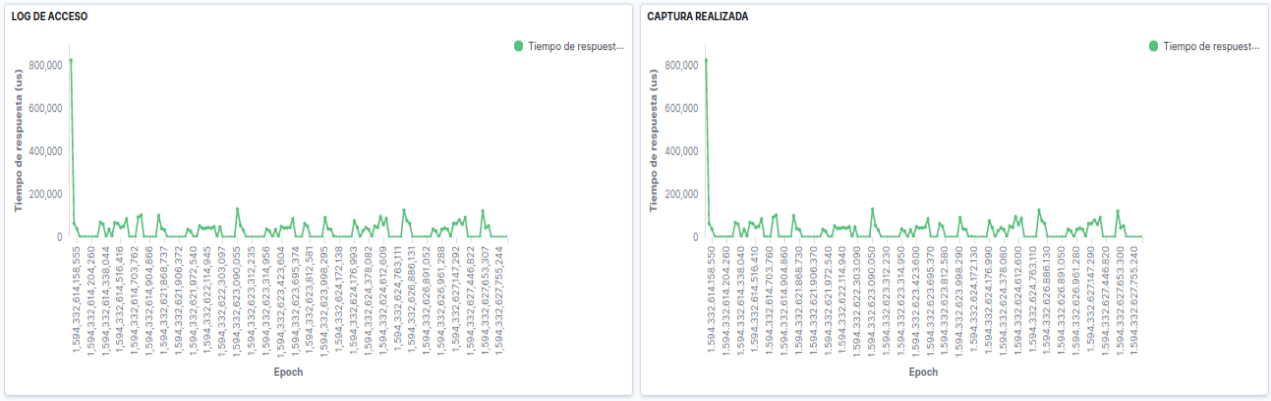

Figura 6-49: Dashboard de Kibana para monitorizar los tiempos de respuesta de la base de datos de tráfico ideal del servidor HTTPS

- Resultados sobre la base de datos: Tráfico con retardos y limitación en ancho de banda

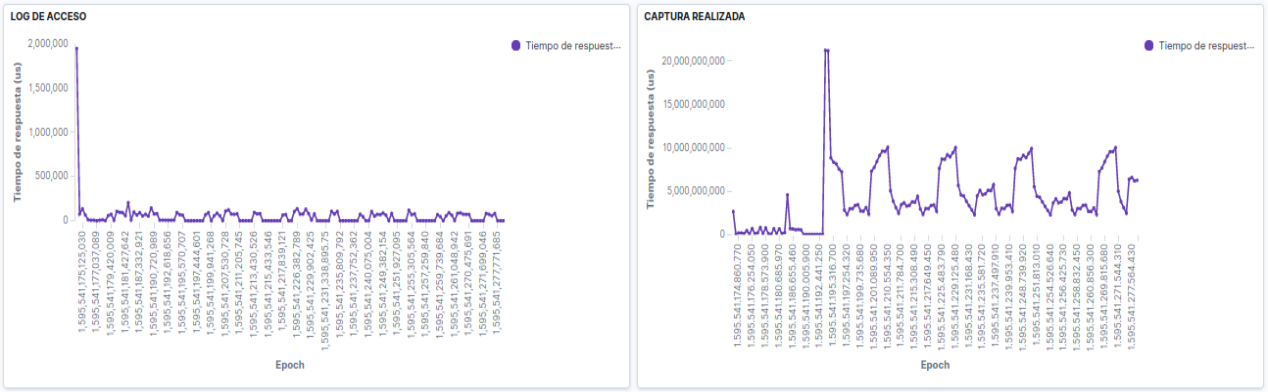

Figura 6-50: Dashboard de Kibana para monitorizar los tiempos de respuesta de la base de datos de tráfico con retardos y limitación en ancho de banda del servidor HTTPS

6.6. Conclusiones

En este capítulo se han presentado todas las pruebas realizadas sobre cada base de datos. Como se comentó en el capítulo 5, esta fase se ha ido realizando en conjunto con la fase de desarrollo. Según se iban realizando las pruebas, se iban realizando modificaciones y mejoras sobre el algoritmo y el sistema, por ejemplo, algunos errores detectados han sido por errores en los formatos en que se envían los datos a Elasticsearch o cuando en ocasiones no se calculaban correctamente los tiempos de respuesta y salían número extraños.

Se ha podido observar que en los casos de tráfico ideal para ambos servidores los tiempos calculados son muy similares, a excepción de la herramienta TSTAT que mide el tiempo de respuesta entre el primer paquete de la petición y el primer paquete de la respuesta. En cuanto a la diferenciación de objetos estáticos y dinámicos, los tiempos de respuesta de los estáticos, al ser tan pequeños, se puede observar mayor diferencia entre las herramientas probablemente por la sincronización de relojes del servidor y de Wireshark, y también que el servidor Apache no es capaz de registrar y escribir tan rápido en el *log* de acceso.

En cuanto al conjunto de herramientas ELK, resulta bastante útil para realizar un seguimiento más exhaustivo y viendo en cada momento que tiempos de respuesta registra cada herramienta.

También, en cuanto se han añadido problemas en la red, hemos podido observar la gran variación en los tiempos de respuesta, llegando el cliente a observar unos tiempos muy diferentes a los que observa el propio servidor.

Por último, podemos concluir que los resultados son buenos e interesantes de cara a que se han encontrado aportaciones que parecían triviales y no lo eran, como que cada herramienta calcula los tiempos de una manera y no siempre coinciden con las del *log* de acceso. Con esto podemos pasar a las conclusiones finales, aportaciones al proyecto y aportaciones para mejorar este proyecto en un trabajo futuro.

7. Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este proyecto se ha realizado la monitorización sobre un servidor HTTP y otro servidor HTTPS mediante el cálculo de los tiempos de respuesta de cada uno. Sobre HTTP hay gran cantidad de estudios de esta métrica, pero aun así se han visto datos interesantes sobre este tipo de servidores. A su vez, la monitorización sobre el servidor HTTPS ha sido más compleja dado que el tráfico está cifrado, pero se han conseguido buenos datos relativos a los tiempos de respuesta, aunque aún hay margen de mejora.

Con la realización de la fase del estudio del arte, se ha entendido la necesidad de realizar esta comparación de los tiempos de respuesta, ya que no resulta tan evidente que las herramientas realicen estos cálculos de manera diferente a la que calcula el servidor. También ha sido de gran ayuda el estudio de la parte de estadística porque no solamente se aplica a este caso en concreto, sino que nos puede servir para realizar otras muchas medidas de comparación de datos en otros ámbitos.

En este punto y entendidas las fases anteriores, se ha podido desarrollar los algoritmos sobre ambos servidores y se ha podido realizar su validación a través de diversas pruebas y bases de datos generadas. Se ha visto que al añadir problemas en la red la correlación entre tiempos de respuesta es más difícil de ver, ya que se generan pérdidas y retransmisiones y hacen que los tiempos varíen mucho.

Por último, para este proyecto han sido necesarios los conocimientos de las asignaturas como Planificación de Redes, Gestión de Redes y Tecnologías y Servicios de Internet. Sin el conocimiento de ellas, la complejidad para realizar este proyecto habría sido mayor.

7.2. Aportaciones al Proyecto

En esta sección se detallan las principales aportaciones obtenidas en este proyecto. Como se ha podido observar durante el Capítulo 6, al modificar la red y añadirle problemas de pérdidas de paquetes y limitaciones en el ancho de banda, lo que registra el servidor no se corresponde con lo que está viendo el cliente. Mientras que el cliente percibe que la página web tarda más de lo normal en cargarse, el servidor es ajeno a esta situación y solamente registra las peticiones y su tiempo de respuesta que observa, pero no llega a registrar las peticiones que el cliente retransmite.

Otra aportación realizada es la manera en la que las herramientas utilizadas calculan los tiempos de respuesta, uno suele pensar que los tiempos que aparecen en dichas herramientas deberían de coincidir con los tiempos que registra el servidor, pero hemos visto que no tiene porqué ser así, y la herramienta TSTAT es un claro ejemplo de ello, ya que registra el tiempo de envío del primer paquete de la petición y el tiempo del envío del primer paquete de la respuesta, por lo que al calcular el tiempo de respuesta será el tiempo entre el primer paquete de la petición y el primero de la respuesta. Mientras que el servidor

Apache y TRANSUM calculan los tiempos de respuesta entre el primer paquete que envía la petición y el último paquete que envía la respuesta.

7.3. Trabajo Futuro

Para este proyecto se proponen las siguientes posibles mejoras, de manera que se puedan conseguir más estadísticas y una mejor monitorización de protocolos de la web:

- Añadir la generación de diversas peticiones extra sobre el servidor como peticiones PUT, POST, DELETE, etc.
- Analizar los tiempos de respuesta que se producen en el protocolo QUIC.
- Realizar la medición del tiempo de respuesta que tarda en cargarse la página web completa, esto es, incluyendo HTML más todos los objetos que contenga la propia página (CSS, javascript, imágenes, etc.). Con esta medida se obtiene el tiempo real percibido por parte del cliente y sería interesante comparar dicho tiempo con el que registra el *log* de acceso del servidor.
- Generar más bases de datos distintas, no solamente incluyendo modificaciones en la red sino haciendo que el servidor tenga limitaciones. Modificando el fichero de configuración del servidor se pueden añadir o modificar los parámetros por defecto del propio servidor.
- Como modo de validación de los algoritmos, se propone validarlo con un *dataset* público o con un servidor real. Para su validación, es necesario disponer del *log* de acceso del servidor y una captura de la traza de red correspondiente.
- Se pueden utilizar otras herramientas de monitorización aparte de las utilizadas en este proyecto. Se han buscado y utilizado las herramientas óptimas para conseguir una buena monitorización sobre los servidores, pero seguramente otras herramientas puedan obtener diferentes parámetros que hagan que se controle más la monitorización sobre ellos.

8. Referencias

[1] Carlos López, Daniel Morato, Eduardo Magaña and Mikel Izal. “Effective Analysis of Secure Web Response Time”. In 2019 Network Traffic Measurement and Analysis Conference (TMA), pp. 145-152. IEEE, 2019.

[2] Thiam Kian Chiew and Karen Renaud. “Estimating web page response time based on server access *log*”. In 2015 9th Malaysian Software Engineering Conference (MySEC), 2014.

[3] Mehul Nalin Vora and Dhaval Shah. “Estimating effective web server response time”. In 2017 Second International Conference on Information Systems Engineering (ICISE). IEEE, April 2017.

[4] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel and Parisa Tabriz. “Measuring HTTPS adoption on the web”. In 26th USENIX Security Symposium, 2017.

[5] R. Fielding and J. Reschke. “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”, 2014. RFC 7231.

[6] E. Rescorla. “HTTP Over TLS”, 2000. RFC 2818.

[7] E. Rescorla. “The Transport Layer Security (TLS) protocol version 1.3”, 2018. RFC 8446.

[8] Informe Actual de Transparencia de Google sobre el Tráfico HTTPS, https://transparencyreport.google.com/https/overview. [Último acceso: 1 de septiembre de 2020].

[9] J. Postel. “Transmission Control Protocol”, September 1981. RFC 793.

[10] J. Postel. “User Datagram Protocol”, August 1980. RFC 768.

[11] J. Iyengar, I. Swett. “QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2”, June 2015, *draft-tsvwg-quic-protocol-00*.

[12] Área de Ingeniería Telemática. “TCP: Ventana de control de flujo y timers”. Gestión y Planificación de Redes y Servicios, Universidad Pública de Navarra.

[13] Mónica Martínez Gómez y Manuel Marí Benlloch, “La Distribución de Poisson”, https://riunet.upv.es/bitstream/handle/10251/7937/Distribucion%20Poisson.pdf, Universidad Politécnica de Valencia. [Último acceso: 1 de septiembre de 2020].

[14] “Modelos de Probabilidad II”, https://www.uv.es/lejarza/eaa/teoria/EAA6-b.pdf, Facultad de Economía, Universidad de Valencia. [Último acceso: 1 de septiembre de 2020].

[15] Douglas C. Montgomery and George C. Runger. “Applied Statistics and Probability for Engineers”. 2003, pp 63-65.

[16] Jorge Ortiz y Álvaro Montenegro. “Modelamiento Estadístico”. Departamento de Estadística, Universidad Nacional de Colombia, pp. 113-114. 2005.

[17] R. García Bellido, J. González Such y J.M. Jornet Meliá. “SPSS: Pruebas no paramétricas”. Convocatoria de Innovación del Vicerectorat de Convèrgencia Europea i Qualitat de la Universitat de València.

[18] S. Kullback and R. A. Leibler. “On information and sufficiency”. The George Washington University and Washington, D.C., 1951.

[19] Sonia Castillo Gutiérrez y Emilio Damián Lozano Aguilera. “Q-Q Plot Normal. Los puntos de posición gráfica”. Departamento de Estadística e Investigación Operativa, Universidad de Jaén, 2007.
[20] “Quantile-Quantile Plot in R”, https://statisticsglobe.com/r-qqplot-qqnorm-qqline-function. [Último acceso: 1 de septiembre de 2020].
[21] Carlos Vega, Paula Roquero and Javier Aracil. “Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams”. *Computer Networks 113*, pp. 258-268, 2017.
[22] Carlos Gonzalo Vega Moreno. “Explorando el proceso de recolección, análisis y visualización del tráfico en las redes de computadoras”. Tesis Doctoral, noviembre 2017, Madrid.
[23] Sistema Operativo Ubuntu, https://ubuntu.com/ [Último acceso: 1 de septiembre de 2020].
[24] Tendencia histórica de Linux, https://w3techs.com/technologies/history_details/os-linux [Último acceso: 1 de septiembre de 2020].
[25] Oracle VM Virtualbox, https://www.virtualbox.org/. [Último acceso: 1 de septiembre de 2020].
[26] Servidor Web Apache HTTP, https://httpd.apache.org/. [Último acceso: 1 de septiembre de 2020].
[27] Descripción del Servidor Web Apache, https://www.hostinger.es/tutoriales/que-es-apache/. [Último acceso: 1 de septiembre de 2020].
[28] Wireshark, https://www.wireshark.org/. [Último acceso: 1 de septiembre de 2020].
[29] Herramienta de Análisis de Tráfico Tstat, http://tstat.polito.it/. [Último acceso: 1 de septiembre de 2020].
[30] ELK Stack, https://www.elastic.co/es/elk-stack. [Último acceso: 1 de septiembre de 2020].
[31] Jesús E. Díaz-Verdejo, Antonio Estepa, Rafael Estepa, Germán Madinabeitia y Fco. Javier Muñoz-Calle. “A methodology for conducting efficient sanitization of HTTP training datasets”. Future Generation Computer Systems, pages 67-82, 2020.
[32] Linux Traffic Control, https://linux.die.net/man/8/tc. [Último acceso: 1 de septiembre de 2020].