Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Trabajo Fin de Grado

DESARROLLO DE UN DISECTOR DE TRÁFICO PARA EL PROTOCOLO QUIC

Autor: Víctor Morales Gómez
Tutor: Jorge Enrique López de Vergara Méndez

NOVIEMBRE 2019

DESARROLLO DE UN DISECTOR DE TRÁFICO PARA EL PROTOCOLO QUIC

Autor: Víctor Morales Gómez
Tutor: Jorge Enrique López de Vergara Méndez

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
NOVIEMBRE 2019

Resumen

Desde los años 80, la capa de transporte en las redes de Internet ha sido controlada principalmente por dos protocolos: TCP (Protocolo de Control de Transmisiones, *Transmission Control Protocol*) y UDP (Protocolo de Datagramas de Usuario, *User Datagram Protocol*); el primero orientado a las conexiones mientras que el segundo al envío de datagramas. Ambos protocolos presentan dificultades para adaptarse a los nuevos requerimientos de las redes actuales. Por ello, Google ha desarrollado un nuevo protocolo, QUIC (Conexiones UDP Rápidas en Internet, *Quick UDP Internet Connections*). QUIC fue presentado en 2014 y actualmente se usa en el navegador web Google Chrome, así como en el resto de servicios que ofrece el ecosistema de Google. Este protocolo posee la cualidad de usar UDP como protocolo de transporte, pero añade las características de control y cifrado de conexiones correspondientes a TCP y TLS (Seguridad de la Capa de Transporte,*Transport Layer Security*) respectivamente.

En este Trabajo de Fin de Grado se ha desarrollado un disector de tráfico red. El objetivo es analizar los paquetes con contenido QUIC y obtener información de interés de las sesiones que hagan uso de dicho protocolo. De esta manera, es posible monitorizar el estado de este tipo de conexiones en un segmento de red, lo cuál es útil para detectar problemas en los servicios. Para asegurar que los resultados son correctos, se han llevado a cabo pruebas en tres entornos web diferentes. Finalmente, se han comparado los resultados con los obtenidos por Wireshark, el disector de referencia, validando de esta manera la funcionalidad del desarrollo realizado.

Palabras Clave

Análisis, TCP, TLS, UDP, QUIC, datagrama, cifrado, Google, Google Chrome, monitorización de red, seguimiento de sesiones.

Abstract

Since the 80s, the transport layer of the Internet has been mainly controlled by two protocols: TCP (*Transmission Control Protocol*) and UDP (*User Datagram Protocol*); the first one is guided to connection control whether the second one is focused on datagram transmission. Both of these protocols struggle to adapt to the new requiriments of the net. Therefore, Google has developed the newer protocol, QUIC (*Quick UDP Internet Connections*). QUIC was presented by 2014 and nowadays is used in the web browser Google Chrome, as well as the rest of the services provided by the Google ecosystem. This protocol has the attribute of using UDP as transport layer however, it adds the connection control and encryption qualities of TCP and TLS (*Transport Layer Security*) respectively.

In this Project a web traffic dissector has been developed. The main objective is to analyze the content of QUIC packets and obtain significant information of the sessions that make use of this protocol. By this way, its possible to monitor the state of this type of connections in a segment of the net, this is useful to detect problems in the services. To ensure that the results are accurate, it has been tested in three different web enviroments. Finally, the results have been compared with the results obtained by Wireshark, the packet dissector of reference.

Keywords

Analysis, TCP, TLS, UDP, QUIC, datagram, encryption, Google, Google Chrome, web monitoring, session tracing.

Agradecimientos

Me gustaría comenzar agradeciendo a mis padres. A Silvia, mi madre, por ser una fuente inagotable de amor y cariño, así como un excelente ejemplo de trabajo y dedicación. También a mi padre, Antonio, porque un día decidiste regalarme un libro, El Diablo de los Números, sin saber que despertaría en mí una gran pasión por las matemáticas. Eres mi ejemplo a seguir y, aunque no siempre estemos de acuerdo, agradezco todos y cada uno de los consejos que me proporcionas.

También quiero agradecer a mi hermano Carlos, que siguiendo los pasos de esta familia has decidido estudiar Teleco, eres una mente prodigiosa y te espera un futuro brillante. No me olvido de Dexter, nuestro perro, a él también quiero agradecerle el amor incondicional que siempre nos aporta.

A su vez deseo agradecer a todos mis amigos el tiempo que pasamos juntos. Y también a mis compañeros de la carrera. Que juntos hemos sobrevivido a los exámenes de CAP, Ondas y Antenas. En concreto, quiero agradecer a aquellos con quienes sé que voy a mantener una amistad duradera: Patri, Adri, David, Ferchu y a todo el Agrupamiento Primario.

Finalmente, quiero agradecer a todos los profesionales de la comunidad universitaria por el gran trabajo que realizan. Al grupo HPCN y a mi tutor, Jorge E. López de Vergara, quien me propuso este trabajo y de quien he aprendido muchísimo. Le considero uno de los mejores tutores que he tenido en mi vida.

¡Muchas gracias a todos!

Víctor Morales Gómez

Noviembre 2019

Índice general

Índice de Figuras

Índice de Tablas

Glosario de acrónimos

- **ACK**: *Acknowledgement.*
- **ASCII**: *American Standard Code for Information Interchange.*
- **CHLO**: *Client Hello.*
- **CID**: *Connection Identification.*
- **FEC**: *Forward Error Correction.*
- **GQUIC**: *Google Quick UDP Internet Connections.*
- **HTTP**: *Hypertext Transfer Protocol.*
- **IETF**: *Internet Engineering Task Force.*
- **IP**: *Internet Protocol.*
- **OSI**: *Open System Interconnection.*
- **PKN**: *Packet Number.*
- **QOS**: *Quality Of Service.*
- **QUIC**: *Quick UDP Internet Connections.*
- **REJ**: *Rejection.*
- **RTT**: *Round Trip Time.*
- **TCP**: *Transmission Control Protocol.*
- **TLS**: *Transport Layer Security.*
- **UDP**: *User Datagram Protocol.*

1 Introducción

1.1. Motivación

Aunque Internet ha sido empleado para la transmisión de contenido web, en los últimos años ha habido una drástica evolución de las tecnologías y los servicios ofrecidos. Algunas de las nuevas tecnologías son: 5G, dispositivos móviles, Internet of Things o virtualización. A su vez se han desarrollado nuevas aplicaciones: streaming de contenido multimedia, vídeo bajo demanda, Big Data, redes sociales y servicios a tiempo real. Estos nuevos servicios son cada vez más demandantes de recursos y requieren de mayor protección de la información. Aplicaciones como Youtube utilizan actualmente más de un 10 % del tráfico de red. Y este porcentaje va en aumento [4]. Pero, sin duda, la aplicación que más tráfico conduce es Netflix, con entre un 15 % y un 40 % en hora punta [5].

Actualmente, Youtube es propiedad de Alphabet, parte de la matriz de Google LCC. El gigante de los servicios de Internet tiene muy presente la importancia de optimizar su uso de red. Por ello, ha desarrollado QUIC (*Quick UDP Internet Connections*, Conexiones UDP Rápidas en Internet), un nuevo protocolo de red cuyo objetivo es optimizar los recursos de la red, disminuir la latencia, aumentar el ancho de banda y añadir más seguridad a las comunicaciones.

Para ello, el protocolo QUIC usa como capa de transporte al protocolo clásico de Internet UDP (*User Datagram Protocol*, Protocolo de Datagramas de Usuario). UDP ofrece algunas ventajas como baja latencia y uso mínimo de datos de cabecera. Pero tiene una desventaja frente a su principal competidor, TCP (*Transmission Control Protocol*, Protocolo de Control de Transmisiones) la falta de transporte fiable de los datagramas. Esto quiere decir que, en caso de perder un paquete de datos en UDP, este no se podría recuperar. Esto no ocurre en TCP, gracias a sus mecanismos de control de pérdidas y reenvío de paquetes.

El protocolo QUIC fue desarrollado en 2014 y comenzó su uso en 2015. En 2017, el protocolo QUIC se utilizaba en más de un 15 % de las conexiones de Internet [6]. Se ha convertido en muy poco tiempo en uno de los principales protocolos de la red y es por

ello que se vuelve cada vez más necesario tener herramientas para poder analizarlo. Es el protocolo preferido en HTTP/2 (Hypertext Transfer Protocol, Protocolo de Transferencia de Hipertexto) y en el futuro HTTP/3.

1.2. Objetivos

Una de las dificultades principales del estudio de este protocolo es que, debido a su reciente implementación y sus frecuentes actualizaciones no existen muchos estudios de su funcionamiento o rendimiento. Por ello, los objetivos propuestos a cumplir en este Trabajo de Fin de Grado son el estudio y comprensión del funcionamiento del protocolo. Posteriormente, se propone el diseño de un disector de paquetes para la obtención de los campos relevantes de cada datagrama, recopilación de la información de una sesión y obtención de las medidas de rendimiento del protocolo. Este desarrollo se llevará a cabo en C, para alcanzar altas tasas de procesamiento, cercanas a los 10 Gbit/s. Y se mostrarán los resultados obtenidos por sesiones en vez de por paquetes, de esta forma, se podrán detectar anomalías en la red.

Para asegurar que los resultados son correctos se comprobarán con los ofrecidos por el disector de Wireshark. Esta herramienta es un analizador de redes muy potente del que hablaremos más adelante.

1.3. Fases de realización

La realización de este trabajo se ha dividido en las siguientes etapas, como se puede ver en el Diagrama de Gantt de la **Figura 1.1**.

- **Estudio del estado del arte:** Durante este periodo se recopiló y estudió la información disponible del protocolo. Primero se estudió el funcionamiento de TCP y UDP, ya que ambos se usan como base del desarrollo de QUIC. Más tarde, se dio paso al estudio de QUIC. Para ello, se consultaron estudios previos de otros grupos de investigación. También se estudiaron los recursos ofrecidos por los desarrolladores del protocolo, en concreto la web de Chromium y la IETF (*Internet Engineering Task Force*, Grupo de Trabajo de Ingeniería de Internet). También se comenzó a usar Wireshark para analizar tráfico web.

- **Diseño y Desarrollo:** En esta parte se implementó el disector de tráfico QUIC. Para ello se implementó un disector de IP (Internet Protocol, Protocolo de Internet)/TCP/UDP que se usaría como base para el desarrollo de la versión QUIC. Más tarde se desarrolló el disector protocolo completo. Tras su finalización se llevaron a cabo tareas de optimización y mejoras al disector para que el funcionamiento fuera lo mejor posible.

- **Pruebas y validación:** Para asegurar que los resultados del disector eran correctos se llevaron a cabo distintas validaciones. Se utilizaron trazas de Internet de distintas características y se realizó un script que detectaba posibles errores para su corrección.

- **Escritura de la memoria:** Por último, se llevó a cabo la redacción de la memoria del Trabajo de Fin de Grado. En ella se describe todo lo aprendido durante este proceso y la información relevante en caso de que en un futuro se decida proseguir la investigación de este protocolo.

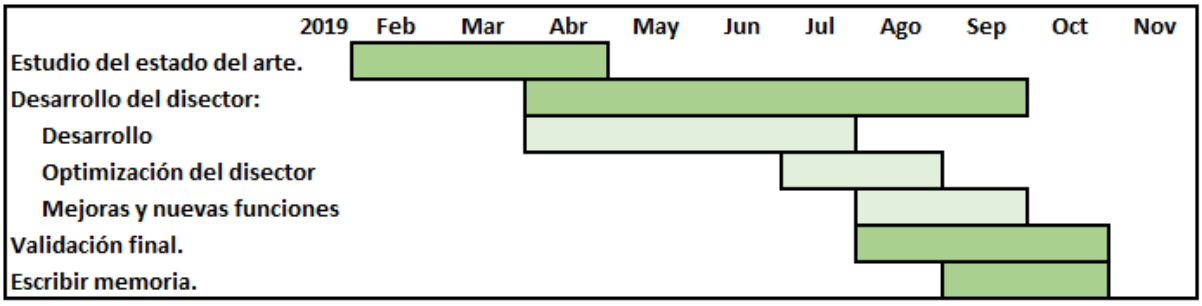

Figura 1.1: Diagrama de Gantt.

1.4. Estructura del documento

El resto de la memoria consta de los siguientes capítulos:

- **Estado del Arte:** En este capítulo se va a explicar de forma general el funcionamiento de los protocolos TCP y UDP. Después, se profundizará en el funcionamiento del protocolo QUIC. Se comentarán las características de la biblioteca software LibPCAP, necesaria para el desarrollo del disector. Por último, se llevará a cabo una explicación del funcionamiento de los disectores de tráfico y se tomará como ejemplo Wireshark y su versión por línea de comandos, Tshark.
- **Análisis de Requisitos:** En esta parte se definirán los requisitos que debe cumplir el disector para poder llevar a cabo su funcionamiento de forma satisfactoria. Se ha dividido en requisitos funcionales y no funcionales.
- **Diseño y Desarrollo:** Primero se hablará del entorno de desarrollo usado y el por qué de su utilización. Luego se procederá a describir el desarrollo del disector de protocolo QUIC usando los requisitos como objetivo a cumplir. Se hablará de las distintas fases llevadas a cabo durante la implementación, incluyendo la optimización y mejora del diseño.
- **Resultados:** En este capítulo se mostrarán los resultados obtenidos por el disector usando distintas capturas de tráfico de red. Cada captura se ha realizado usando un servicio de Internet distinto para poder comprobar cuál es el funcionamiento del protocolo QUIC en distintos escenarios. Después se comparan los datos obtenidos por los distintos resultados y por último, se documenta el comportamiento del disector frente a protocolos distintos a QUIC que puedan aparecer.
- **Validación:** Para comprobar que los resultados obtenidos son fiables, se ha realizado una validación que queda explicada en este apartado. Se detallarán las distintas pruebas realizadas, sus resultados y los cambios realizados al disector según los fallos encontrados. También se comenta el rendimiento obtenido por el disector.

- **Conclusiones y Trabajo Futuro:** Por último, se explicará las conclusiones finales obtenidas al terminar el desarrollo del Trabajo de Fin de Grado. También se identificarán nuevas líneas de investigación basadas en el disector de tráfico QUIC y posibles trabajos según los resultados obtenidos del análisis de protocolo QUIC.

2

Estado del Arte

2.1. Introducción

En este apartado se expondrán los conocimientos previos necesarios a la realización del proyecto. Se pondrá en contexto la situación en la que se encuentran TCP y UDP, sus características y limitaciones. Luego se profundizará en el protocolo QUIC. Primero se comentarán los campos de las cabeceras y después los mecanismos de control que lleva a cabo cada protocolo. Finalmente, se hará una descripción de los disectores de paquetes y en concreto de Wireshark y Tshark.

2.2. QUIC

QUIC es un protocolo de red diseñado por Google para la capa de transporte. Está basado en UDP y tiene como objetivo proveer un servicio mejor que TCP + TLS (Transport Layer Security, Seguridad de la Capa de Transporte). A continuación, veremos las características de estos protocolos de red. Al ser un protocolo experimental, tiene actualizaciones cada pocos meses. Esto hace que cada poco tiempo los campos de las cabeceras cambien y podrían no ser iguales a los descritos a continuación. Para este Trabajo de Fin de Grado se ha estudiado la versión GQUIC (Google Quick UDP Internet Connections, Conexiones UDP Rápidas en Internet de Google) Q043, que es la que implementa Google Chrome actualmente (Versión 77.0.03865.90). Para comprobar la posición de los campos se han analizado paquetes capturados previamente en Wireshark.

2.2.1. TCP y UDP

La idea de este apartado es tener una vista general de los protocolos TCP y UDP en relación al protocolo QUIC, no un estudio a fondo de su funcionamiento.

El protocolo de red TCP fue diseñado en los años 80 y ha demostrado su utilidad como principal protocolo de transporte de paquetes en la red. Sus características de control de conexiones lo hacen el favorito para el envío de información de forma fiable. Algunas de sus características son:

- **Establecimiento de conexión:** El establecimiento de conexión o *Handshake* permite al cliente establecer una conexión con el servidor. Requiere de tres pasos: SYN, SYN-ACK y ACK (Asentimiento, Reconocimiento). Cuando se establece conexión satisfactoriamente, se procede a la transmisión de la información. Este sistema permite la negociación del número de secuencia por parte del cliente y el servidor antes del envío de información, también es posible negociar otros parámetros pero serían opcionales. El proceso de establecimiento de conexión se lleva a cabo a cambio del tiempo usado en el envío y recepción de los tres paquetes, lo que aumenta considerablemente la latencia de la transmisión y resulta poco eficiente en conexiones breves.
- **Números de secuencia y asentimientos:** Para asegurar que los datos llegan ordenados TCP implementa los números de secuencia, empezando en un número aleatorio, para garantizar la seguridad. Cada intercambio de datos en el flujo aumentará el número de secuencia del receptor. Una mejora de este sistema es SACK, que permite asentir los datos recibidos, de forma que, si se pierde información en la transmisión, el emisor puede saber exactamente qué segmento se ha extraviado y reenviarlo.
- **Ventanas deslizantes:** Para evitar que el emisor mande más datos de los que puede soportar la red o el receptor, durante el establecimiento de conexión se negocia una cantidad máxima de bytes que se pueden enviar. Esta cantidad puede variar durante la conexión pero siempre a petición del receptor.

Las propiedades comentadas permiten a TCP asegurar una comunicación fiable. Pero cada vez son mayores sus limitaciones para dar respuesta a los desafíos que supone al Internet actual.

Por ejemplo, TCP no incorpora ningún tipo de cifrado de datos. Es por ello que requiere de ser usado junto con TLS para garantizar la protección de la información. A cambio, cada paquete de información tendrá una latencia mayor y una cabecera de datos adicional. En la **Figura 2.1** (Campos TLS 1.2 no incluídos) se puede observar cómo está formada la cabecera TCP.

Bytes	0								1								2								3							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	Puerto Origen																Puerto Destino															
4	Número de secuencia																															
8	Número de asentimiento																															
12	Long. Cab.				Reservado							Banderas					Ventana															
16	Suma de verificación																Puntero de urgencia															
20	Opciones (si las hay)																								Relleno							
...	Data																															

Figura 2.1: Cabecera TCP

Por otra parte, está UDP. Este protocolo también fue desarrollado en los años 80 y tiene una aproximación distinta a TCP en el manejo de conexiones. Si bien TCP tenía como objetivo asegurar la conexión fiable, UDP no asegura que todos los paquetes enviados lleguen a su destino. Esto no es especialmente grave en aplicaciones como el *streaming* de audio o vídeo. En estos casos unas bajas pérdidas a cambio de menor latencia y mayor ancho de banda puede ser preferible. UDP tampoco implementa protección de la información transportada, para subsanar dicho problema hace uso del protocolo DTLS (*Datagram TLS*).

UDP no implementa establecimientos de conexión ni control de flujo o congestión, únicamente una suma de verificación para asegurar que el contenido de un paquete no ha sido corrompido en su transcurso por la red. En la **Figura 2.2** se puede observar cómo está formada la cabecera UDP.

Bytes	0								1								2								3							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	Puerto Origen																Puerto Destino															
4	Longitud																Checksum															
8 ...	Data																															

Figura 2.2: Cabecera UDP

En lo que respecta a TLS, fue diseñado para proveer comunicaciones seguras en las redes de ordenadores. Funciona mediante la encriptación de la información con claves simétricas secretas compartidas entre el cliente y el servidor. DTLS sería la versión que provee seguridad a los datagramas de UDP. Para alcanzar una calidad de seguridad aceptable, el cliente y el servidor intercambian los algoritmos de cifrados de los que disponen, en lo que se conoce como *TLS handshake*, este proceso, añade aún más tiempo de establecimiento de conexión, aumentando con ello la latencia.

Para finalizar la sección se puede observar la **Tabla 2.1** resumen comparando ambos protocolos.

Tabla 2.1: TCP y UDP

	TCP	UDP
Nombre	Transport Control Protocol	User Datagram Protocol
Tamaño de cabecera	20 Bytes	8 Bytes
Latencia	Alta debido al Handshake y Ventanas de conexión	Baja. Envío a la máxima tasa permitida sin verificación previa.
Confiabilidad	Garantizada. Uso de ACKs y retransmisiones	No garantizada. Se puede perder información
Cifrado	No. Usa TLS para cifrar datos	No. Usa DTLS para cifrar datos.
Aplicaciones	HTTP, FTP, SMTP	DNS, VoIP

2.2.2. Google y QUIC

QUIC es el nombre que recibe el protocolo de red en el que Google ha estado trabajando desde 2014. En 2015 lanzó la primera versión para Chromium y más adelante para Google Chrome, el navegador web más popular. Desde entonces se ha expandido su uso a todos los servicios que ofrece Google.

Como se ha comentado anteriormente, QUIC trabaja sobre el protocolo UDP y comparte las funcionalidades de control de conexiones que tiene TCP. También implementa las funcionalidades de cifrado de datos propias de TLS. La **Figura 2.3** ayudará a visualizarlo.

Se puede observar una pila de protocolos, según el modelo OSI (*Open System Interconnection*, Interconexión de Sistemas Abiertos). Esta representación sería una abstracción de la funcionalidad que proporciona cada protocolo según la capa en la que se encuentra. Lo que se pretende ilustrar es que si bien en TCP + TLS las capas de transporte y seguridad quedan bien diferenciadas por protocolo. En UDP + QUIC no existe dicha diferencia. Esto se debe a que el protocolo incorpora los elementos de control de conexión, correspondientes a la capa de transporte, junto con elementos de cifrado de los datos, correspondiente a la capa de seguridad.

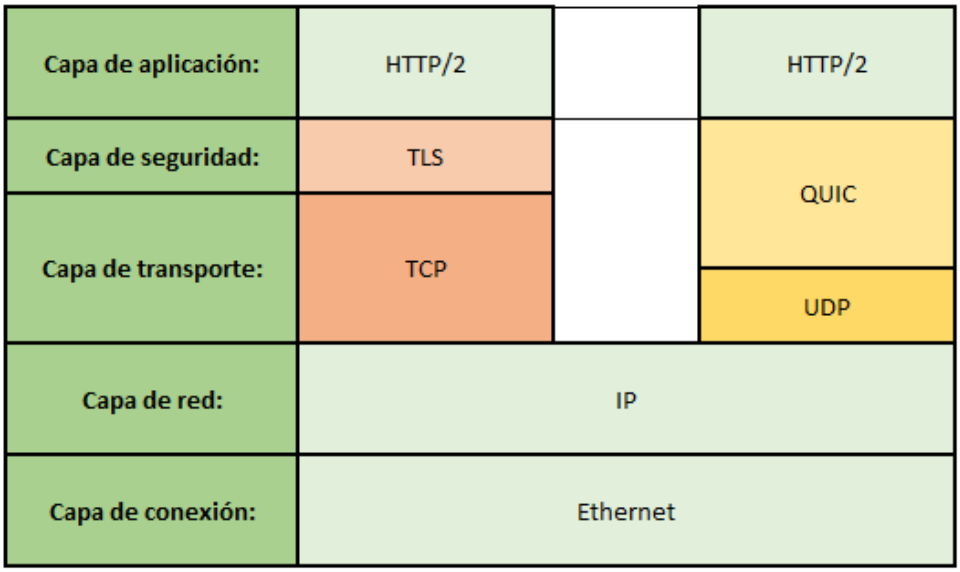

Figura 2.3: Pila de protocolos, modelo OSI

2.2.3. Campos de la trama

Si bien QUIC es la definición formal que existe del protocolo y las características que debe tener, GQUIC sería la versión implementada en el buscador Google Chrome. Una de las peculiaridades del protocolo GQUIC es que, a diferencia de un paquete de TCP, los paquetes de GQUIC pueden variar la cantidad de campos que envían en un paquete [7]. Además, los campos de estos paquetes pueden variar de tamaño. De esta forma, y como se observa en la **Figura 2.5**, los primeros paquetes se usarían para el establecimiento de conexión, podrían llevar una cabecera mayor. Mientras que los paquetes que solo se usan para enviar datos tendrían una cabecera con solo el número de secuencia, como se muestra en la **Figura 2.6**. Aún así, todos los paquetes contienen siempre los campos *Public Flags* y PKN (*Packet Number*, Número de Paquete) [8].

Los campos que se usan en cada paquete se pueden ver en el campo de la cabecera *Public Flags* de la **Figura 2.4**. De bit menos significativo a bit más significativo las banderas serían las siguientes:

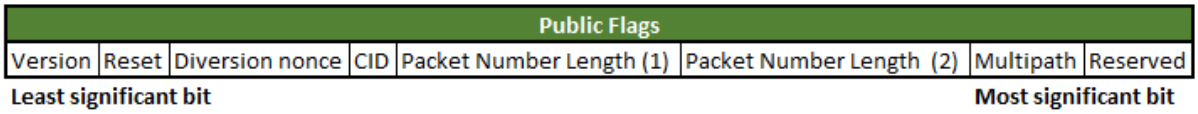

Figura 2.4: Public Flags

- ***Version:*** Indica cuándo aparece el campo *Version*. 1 será SI y 0 será NO. El campo *Version* indica en ASCII (*American Standard Code for Information Interchange*, Código Estadounidense para el Intercambio de Información) que versión del protocolo se está enviando. La más actualizada y la que se ha estudiado, es Q043.
- ***Reset:*** Indica un *reset* público de la sesión. 1 será SI y 0 será NO.
- ***Diversion nonce:*** Indica cuándo aparece el campo *Message Hash Authentication*. 1 será SI y 0 será NO. En caso de que este campo aparezca y no haya CID (*Connection Identification*, Identificador de Conexión) se enviará un *diversification nonce* que es un código aleatorio de 32 Bytes.
- ***CID:*** Indica cuándo aparece el campo CID. 1 será SI y 0 será NO. Este campo se compone de 8 Bytes que indican el número identificador de la sesión. Este código deberías ser único para una sesión en todo momento.
- ***Packet Number Length:*** Indica cuánto medirá el campo *Packet Number*. Si es 00 el campo medirá 1 Byte, si es 01 medirá 2 Bytes, si es 10 medirá 4 Bytes y si es 11 medirá 6 Bytes. En caso de usar 6 Bytes el rango de números posibles sería de 1 a 2^{48} ($2.8x10^{14}$) paquetes por conexión. El campo PKN sería el equivalente al número de secuencia en TCP. La principal diferencia es que mientras que en TCP se comenzaba por un número aleatorio, en GQUIC el PKN empieza en 0 y va aumentando de uno en uno hasta llegar al máximo definido en el *Public Flags*. En ese caso pueden ocurrir dos cosas, que el *Public Flag* actualice el máximo PKN posible y por lo tanto el PKN seguiría aumentando con cada paquete enviado o podría no actualizarse el campo y truncar el número. De esta manera volvería a enviar 0.[8]

- ***Multipath:*** Indica cuándo una conexión usa múltiples caminos para el envío de los paquetes. 1 será SI y 0 será NO.
- ***Reserved:*** Este campo aún no tiene uso, puede que en un futuro si se utilice. Siempre es 0.

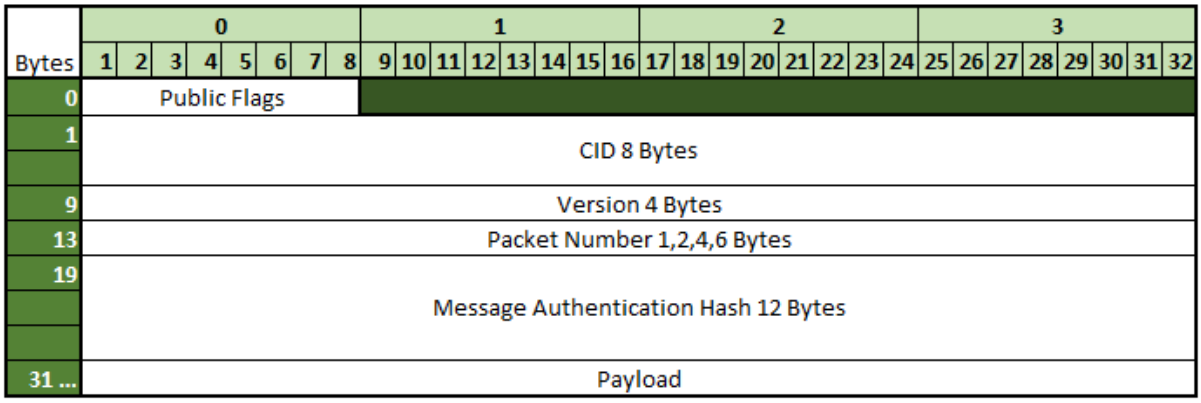

Figura 2.5: Cabecera QUIC extendida

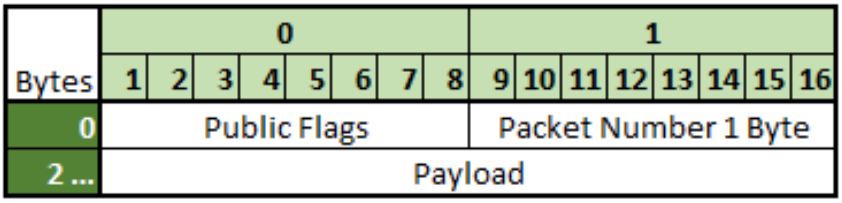

Figura 2.6: Cabecera QUIC mínima

A continuación, se van a comentar los paquetes especiales, estos son aquellos que llevan la información necesaria para el establecimiento de la conexión, control de la misma y acuse de recibo. Los campos del *special packet* vendrían después de los campos públicos vistos anteriormente.

Tendrían el formato de la **Figura 2.8**, los primeros campos indicarían que tipo de paquete es y a continuación, se indicarían los campos TAG, que cada uno describe un tipo información.

Similar al *Public Flag*, los paquetes especiales contienen el campo *SpecialFrameType*, en este se puede ver que campos aparecen a continuación.

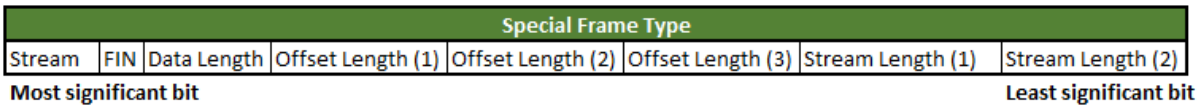

Figura 2.7: Cabecera Special Frame Type

- ***Stream:*** Indica si es un *Stream Special Packet*.
- **FIN:** Indica si es FIN. 1 será YES y 0 será NO.
- ***Data Length:*** A veces los paquetes especiales también envían datos. El campo *Data Length* indica cuánto miden esos datos. 1 será YES y 0 será NO.
- ***Offset:*** Indica cuanto mide el campo *Offset*. Si es 000 el campo medirá 0 Bytes, no aparecerá, si es 001 el campo medirá 1 Byte.
- ***Stream Length:*** Indica el tamaño del campo *StreamID*. Si es 00 el campo medirá 1 Byte.

Fuera del campo *Special Frame Type* podemos encontrar estos otros campos:

- **TAG:** Indica en código ASCII si es un paquete CHLO (*Client Hello*, Saludo del Cliente) o si es un REJ (*Rejection*, Rechazo). Estos tipos de paquetes tendrían un funcionamiento similar al de SYN, SYN-ACK en el establecimiento de conexión de TCP.
- ***TAG number:*** Este campo proporciona el número de TAGs de información que van a aparecer a continuación. Cada campo TAG mide 8 Bytes.
- ***PADDING:*** Relleno de bits, siempre a 0.
- ***TAG Fields:*** Estos campos son muy variados, cada uno proporciona una información distinta. Todos ellos se componen de tres partes.
  - ***TAG Type:*** Es el nombre del campo en ASCII.
  - ***TAG offset end:*** Es el número de bytes que hay que avanzar para encontrar la información del campo.
  - ***TAG value:*** Usando el *TAG offset end*, se puede buscar la información del campo TAG.

En la **Tabla A.1**, se pueden observar los campos TAG que se envían, aunque dependen de cada sesión, cliente y servidor. En este trabajo se ha prestado especial atención en los campos SNI, IRTT, UAID y AEAD, los detalles de dichos campos se muestran en **Tabla 2.2**.

Tabla 2.2: TAGs estudiados

TAG	Nombre (en inglés)	Descripción	Tipo de dato
IRTT	Estimated Initial RTT	Estimación probabilística del RTT. 500 ms por defecto.	Numérico
SNI	Server Name Indication	Indica el Hostname.	String
AEAD	Authenticated Encryption Algorithms	Nombres de los algorítmos de cifrados intercambiados.	String
UAID	Client's User Agent ID	Nombre del cliente que realiza la petición.	String

Bytes	0								1								2								3							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
... N	Special Frame Type								Stream ID																							
N + 2	Offset																Data Length															
N + 6	TAG																															
N + 10	TAG Number																Padding: 0000															
N + 14	TAG Type 1																															TAG
	TAG offset end 1																															Value 1
N + 22	TAG Type 2 ...																															TAG
...	TAG offset end 2																															Value 2

Figura 2.8: Cabecera QUIC Special Packet [1]

Por último, quedarían los paquetes ACK. Los campos de estos paquetes se encuentran entre los públicos y los campos de los paquetes especiales. Pero, siempre acompañados a estos últimos. El formato de la cabecera sería el de **Figura 2.9**. El campo Special Frame Type indica el comienzo de uno de estos paquetes. Si es 0x40 será ACK mientras que si es 0x20 será NACK. Estos paquetes dan la información de que paquetes han sido recibidos.[10]

Bytes	0								1								2								3							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
... N	Special Frame Type								Largest Acked								Largest Acked Delta Time															
N + 4...	First Ack Block Length								Num Timestamp																							

Figura 2.9: Cabecera ACK frame

2.2.4. Sesión QUIC

Como se ha estado comentando, QUIC trabaja sobre UDP, lo que impide tener control sobre la conexión, para poder ser un protocolo orientado a conexiones, como TCP, requiere de algunas funcionalidades que permitan corregir los problemas que puedan surgir durante las transmisión de datos.

- **Establecimiento de conexión:** Al igual que en TCP, QUIC requiere de un establecimiento de conexión, para ello usa los paquetes especiales CHLO (*Client Hello*) y REJ (*Rejection*). Los paquetes CHLO los envía el cliente y mandan al servidor toda la información necesaria para comenzar la comunicación. El servidor, debe responder con un paquete especial REJ, que devolverá al cliente la información requerida por el servidor. Así como las posibles negociaciones de claves y algoritmos. Por lo tanto, QUIC es capaz de realizar el establecimiento de conexión en 1 RTT (*Round Trip Time*, Tiempo de Ida y Vuelta).

  Este intercambio de información es necesario para el establecimiento de conexión. Pero, se puede dar el caso que un cliente y un servidor repitan comunicación en un corto periodo de tiempo. Pensemos en cuando estamos navegando por la web y cerramos una página que a los pocos minutos volvemos a abrir. Para evitar el reenvío de la información de establecimiento de conexión, el cliente y el servidor

guardarán durante un tiempo los datos del otro equipo. De esta forma, al reiniciar una comunicación, se puede empezar a enviar información sin necesidad de un nuevo establecimiento de conexión. Esto se denomina RTT0. Porque la información se transmite con 0 RTTs de latencia. Esto convierte a QUIC, en el protocolo con el establecimiento de conexión más rápido. Por delante de TCP + TLS 1.2 que requieren de al menos un RTT en el caso de reinicio de conexión. El futuro TLS 1.3 añade la función de RTT0, pero aún no está implementado.[11]

En la **Figura 2.10** se muestra la comparación entre los RTT necesarios según el protocolo utilizado. En el caso de QUIC, no se muestra la primera conexión, esta sería equivalente a la de TCP pero teniendo en cuenta que los datos se cifran.

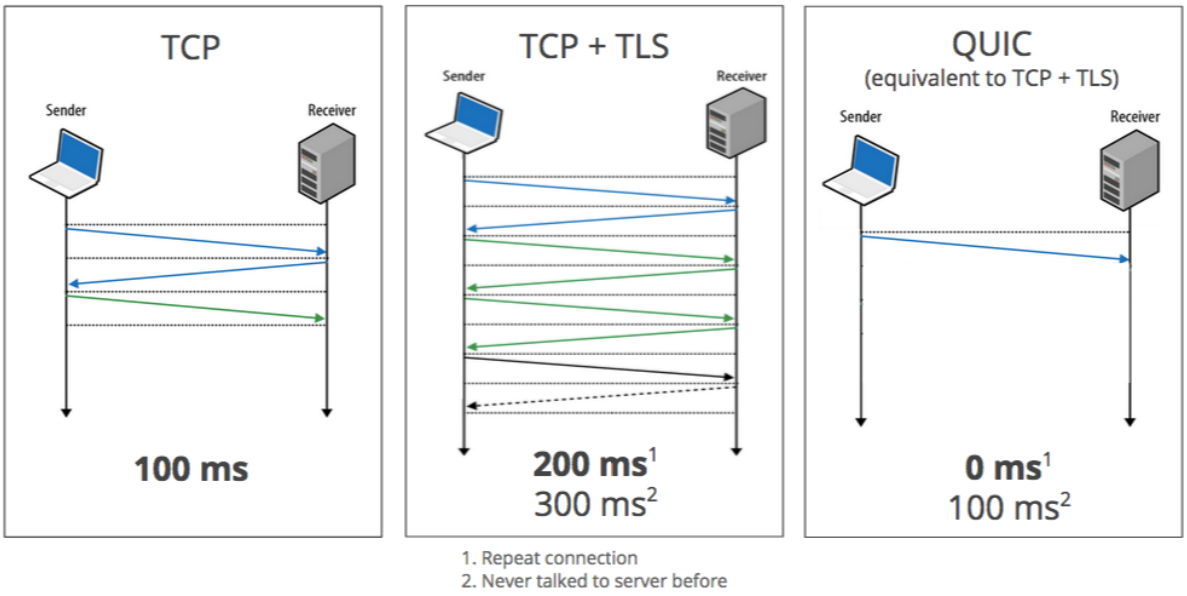

Figura 2.10: Diferencia entre establecimiento en TCP y QUIC [2]

- **Números de secuencia y asentimientos:** El protocolo de red QUIC, tiene la filosofía de enviar el mínimo de paquetes extra posibles. Por lo tanto, asume que todos los paquetes llegan correctamente. Cuando un grupo de paquetes no llega, QUIC debe esperar todo lo posible para garantizar la llegada de esos paquetes, antes de enviar una notificación de último paquete recibido.

- **Ventanas deslizantes:** El cliente y el servidor negocian sus ventanas de envío de datos durante el establecimiento de la conexión, pero también pueden actualizar durante el envío de datos, mandando un *Special Frame*.

- **Multiplexado:** Uno de los problemas de las conexiones web actuales, es que ya no se envía una sola petición de archivos, sino que las cargas webs requieren de múltiples archivos guardados en distintos servidores. Esto, puede aumentar mucho la latencia de la conexión si cada archivo requiere de una nueva conexión con un nuevo establecimiento. Como se observa en **Figura 2.11**, para solucionar este problema, QUIC implementa el sistema de Multiplexado, este sistema permite enviar varias conexiones distintas en un solo paquete. De esta forma, se ahorra en puertos activos y se reduce la latencia total de la conexión. El protocolo SPYD añade este concepto

a las conexiones TCP, pero está limitado por el *Head Of Line Blocking* inherente al protocolo.

El *Head-of-line blocking* puede ocurrir cuando en un protocolo de transporte con mensajes ordenados, pierde uno de estos, y el resto de los mensajes consecutivos deben esperar en la cola hasta que se retransmita el mensaje perdido. Esto provoca que se retrase la conexión. [12]

Para resolver este problema, QUIC envía las distintas conexiones de forma independiente a los otros flujos, de esta forma en caso de perder un paquete con un paquete, únicamente los flujos dependientes de dicho paquete quedarán paralizadas, el resto podrán continuar con la transmisión.

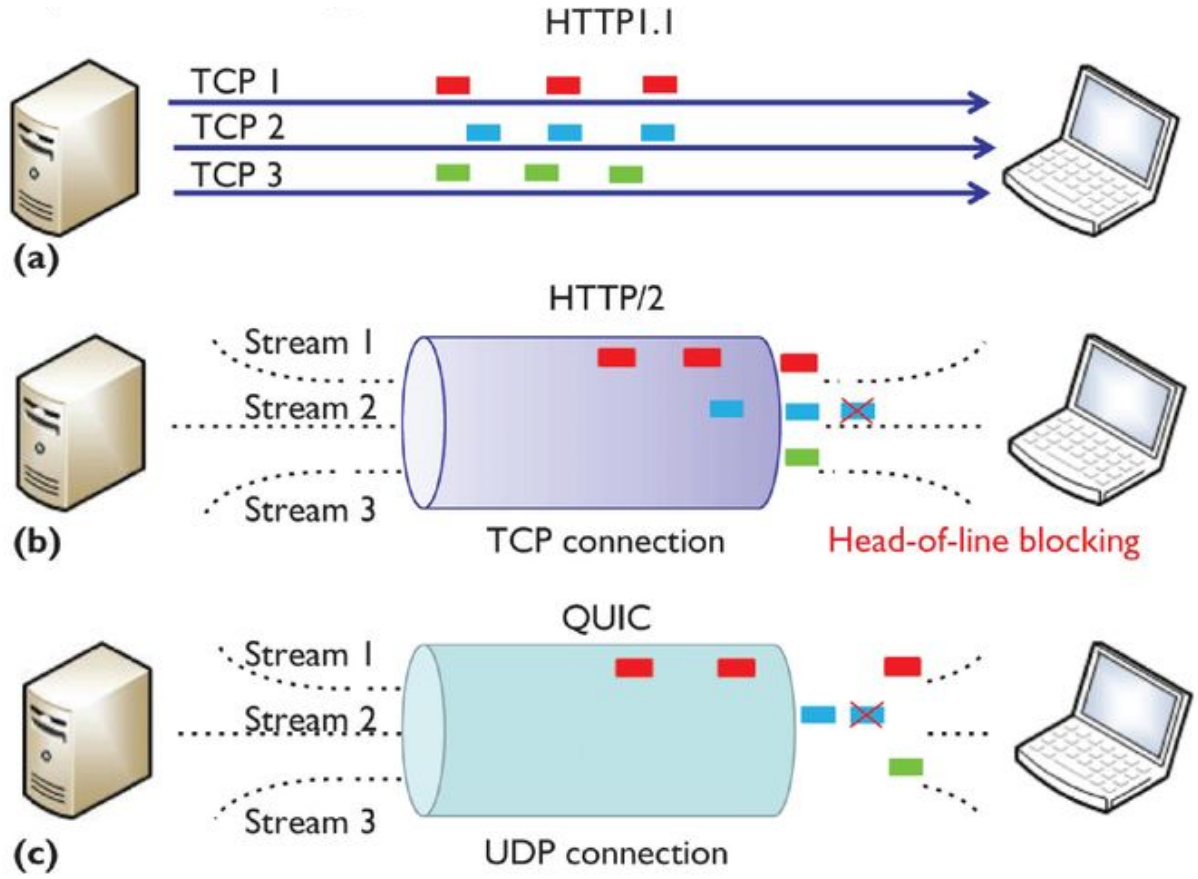

Figura 2.11: Comparación de Multiplexado en TCP y QUIC [3]

- **FEC:** El FEC (Forward Error Correction, Corrección de Errores en Descarga), es el método usado por QUIC para corregir los posibles errores debidos a la pérdida de paquetes en la transmisión. Funciona agrupando varios paquetes en un grupo, a todos los paquetes del grupo se les añade una pequeña información de redundancia. De esta forma, si se pierde un paquete del grupo, se podría recuperar la información de este usando los datos de los otros paquetes. Este sistema no se observa actualmente, probablemente porque las pérdidas en las redes no sean suficientes como para que compense añadir carga extra a cada paquete. Si en cada paquete se enviasen bytes de datos redundantes, esto reduciría la cantidad de bytes de *payload* que se transmite por paquete y por lo tanto, se reduciría el ancho de banda disponible.[13]

- **Cambio de sesión:** En QUIC, es posible que un cliente y servidor cambien de puerto o dirección IP durante una transmisión. Para mantener los datos de la sesión anterior se hace uso del CID, que siempre es único. Esta característica no se ha observado en este trabajo.

- **Cierre de conexión:** Para terminar la conexión, QUIC tiene dos posibilidades. Enviar un paquete especial con el código *CONNECTION CLOSE*, que terminará inmediatamente la conexión. O puede optar por el cerrado en silencio, esto quiere decir que si no se recibe información pasado un tiempo se dará por terminada la conexión. Este tiempo es 30 segundos para ambos extremos de forma predeterminada, pero puede cambiar según la negociación del *Idle State Connection* [9].

2.3. Disector de paquetes

Un disector de paquetes es un programa o equipo hardware que se conecta a una red y es capaz de: capturar una trama, decodificarla, interpretarla y analizarla. Se utilizan para monitorear redes, detectar errores o detectar amenazas a la ciber seguridad.

Los analizadores de protocolo profesionales pueden llegar a ser equipos costosos (+1000€), su precio depende de la velocidad de procesamiento de los equipos y de lo protocolos de red que son capaces de analizar. Por suerte, existen alternativas software gratuitas que están limitadas a la velocidad de procesamiento del ordenador que usemos.

Algunos de los disectores de paquetes que encontramos actualmente son Wireshark y Tshark. Wireshark es un disector de paquetes que incorpora una interfaz gráfica. Esta interfaz permite ver al usuario los resultados de la captura de paquetes a tiempo real. Es completamente gratuito y tiene versiones múltiples plataformas como Windows y Linux. Tambiém es de software libre, por lo que se puede consultar su código. Resultan especialmente útiles sus funciones de filtrado. Permite la captura de paquetes en la red a tiempo real, pero También, permite leer datos almacenados en archivos .pcap, de capturas previas. Es compatible con más de 480 protocolos, incluidos QUIC y GQUIC.

Las limitaciones de Wireshark son: no puede enviar paquetes a la red, únicamente capturar tráfico y procesarlo; no puede modificar los datos capturados y tampoco puede detectar ataques a la red o intrusos.

Tshark es una versión de Wireshark por líneas de comandos. Aunque no tiene una interfaz tan cómoda, como puede ser la de Wireshark, sí que aumenta la velocidad de procesamiento, sobre todo sí se trabaja con trazas de gran tamaño (varios gigabytes).

Los disectores de paquetes comentados anteriormente hacen uso de la biblioteca software libpcap, para el desarrollo de este proyecto también se hará uso de la misma. Libpcap es una librería en C, que permite a los desarrolladores trabajar con paquetes de la capa de enlace directamente, sin tener que preocuparse de las complejidades que se puedan dar por los diferentes sistemas operativos o tarjetas de red. Más adelante, se comentará qué funciones en concreto se han utilizado y cuál es su funcionamiento.

2.4. Conclusión

En este capítulo, se han explicado las bases necesarias para poner en contexto el diseño y desarrollo del disector del protocolo QUIC. En concreto, se ha definido el protocolo QUIC y se ha puesto en relación con los protocolos clásicos de Internet, TCP y UDP. Se ha profundizado en los campos de las cabeceras de los distintos paquetes que tiene el protocolo, porque es importante para el siguiente capítulo entender que, a diferencia de TCP o UDP, en QUIC los campos de las cabeceras, no aparecen siempre y hay múltiples dependencias de unos campos con otros.

Por último, se ha explicado cuál es el funcionamiento de los disectores de paquetes y se ha puesto como ejemplo Wireshark, una aplicación de renombre que se tomará como referencia en el desarrollo del disector.

3

Análisis de Requisitos

3.1. Introducción

Una vez estudiado el funcionamiento teórico del protocolo QUIC, se procede a describir los requisitos que debe cumplir el disector para llevar a cabo su función.

Los requisitos funcionales, serán aquellos que resultan imprescindibles para el funcionamiento del disector. Sin ellos, el comportamiento del sistema será erróneo o deficiente. Los requisitos no funcionales, serán aquellos que aumenten la usabilidad del sistema y permitan al usuario tener una experiencia cómoda usando el programa.

3.2. Requisitos funcionales

El disector de tráfico QUIC, requiere de una serie de requisitos para su correcta implementación.

Debe ser capaz de analizar correctamente el tráfico del protocolo capturado y para cada paquete de la red, se deberá obtener los parámetros, para posteriormente guardarlos. Hay que tener en cuenta, que la implementación de QUIC que hay en Google Chrome es GQUIC Q043.

También, se deberá guardar los datos obtenidos en cada paquete en un registro por sesiones. De esta forma, cada conexión cliente - servidor, tendrá una lista de campos a estudiar posteriormente. Es importante que, cada sesión sea independiente del resto.

Finalmente, deberá implementarse algún sistema de guardado de los datos para el análisis posterior de estos.

3.2.1. Cómo invocar al disector

En esta sección se explica cómo se espera ejecutar el programa. El disector se va a desarrollar en C y se ejecutará por líneas de comando. Para que el usuario pueda saber cómo funciona el disector, se podrá llamar al comando *help*, con ello se mostrará un mensaje con el formato esperado de entrada al programa.

Para llamar al comando help se usará: *./disector -h* o también *./disector + cualquier opción incorrecta*. El formato esperado para el disector es el siguiente: *./disector <-f a.pcap / -i b> [-e "filtro"] [-n x] [-p y]*. A la derecha de cada una de las opciones se pondrá el valor deseado.

Las opciones -f (*file*) y -i (*interface*) son excluyentes, ya que no se puede leer de un archivo y capturar una interfaz de red a la vez. Hay que tener en cuenta que, al leer de un archivo, el programa terminará automáticamente. En caso de capturar una interfaz de red, se deberá abortar el disector cuando se dé por terminada la captura, para ello, se puede hacer uso de Ctrl + C. También, es probable que la opción -i, requiera de permisos de *root*.

La opción -e, invoca al filtro pcap. Es especialmente útil para descartar el tráfico distinto a udp. Pero, en caso de querer añadir un filtro más complejo, se deberá escribir entrecomillado, es la forma que tiene el terminal de Ubuntu de detectar que es un solo valor y no varios valores consecutivos.

Las opciones -e, -n y -p son opcionales, es decir el programa podría funcionar sin necesidad de que estén activadas. Pero, en caso de querer usar las opciones -e y -p se deberá añadir la opción -e también ya que son dependientes.

3.3. Requisitos no funcionales

Los requisitos no funcionales del disector son aquellos que aumentan la usabilidad del sistema. Por ejemplo, el disector debe procesar los paquetes suficientemente rápido como para no perder tráfico de red, aproximadamente 10 Gbit/s. También, es importante que los datos guardados sean fáciles de procesar. Por último, el disector debe mostrar resultados de QOS (*Quality of Service*, Calidad del Servicio) además, de los campos capturados de los paquetes.

Es importante que el disector pueda elegir si se quiere estudiar alguna sesión en concreto. Para ello deberá poder pasar por comandos el filtro deseado.

Finalmente, se deberá realizar una validación de los resultados obtenidos por el disector. Para comprobar si los resultados obtenidos coinciden con los de Wireshark.

3.4. Conclusión

En este capítulo, se han especificado los requisitos que debe cumplir el disector para llevar a cabo su función. Estos requisitos, se usarán para establecer los objetivos a cumplir en el desarrollo y permitirán verificar el cumplimiento de dichos objetivos.

4

Diseño y Desarrollo

4.1. Introducción

En este capítulo, se va a explicar las fases llevadas cabo en el desarrollo del disector de tráfico para el protocolo QUIC. Se usará el análisis de requisitos del capítulo anterior como objetivo a cumplir.

Se va a comenzar explicando cuál ha sido el entorno de desarrollo del programa. Posteriormente, se explicará cómo es la arquitectura del programa y los elementos que lo componen. Finalmente, se comentarán las optimizaciones y mejoras realizadas al programa.

4.2. Entorno de desarrollo

Para el desarrollo de este disector, se ha utilizado una máquina virtual en el programa VMWare. Esta máquina virtual tenía una distribución de Linux, Ubuntu. El principal motivo para usar una máquina virtual es, que nos permite crear un entorno para realizar pruebas con facilidad, tanto para simular redes de ordenadores como para, escribir programas en C, depurar, compilar y ejecutar. También, ayuda el uso de ficheros Makefile y Bash.

Es importante usar una máquina virtual para poder provocar pérdidas de paquetes de red. Para así, poder observar el funcionamiento del protocolo en un entorno realista. VMWare permite editar los ajustes NAT de la máquina virtual en la opción *advanced* de *Network Adapter*, desde ahí se puede poner límite de ancho de banda y porcentaje de paquetes perdidos. Por último, se ha usado la biblioteca Libpcap y la herramienta Tshark, ambas pueden ser instaladas desde un terminal. Se debe usar el comando: *apt-get install libpcap*.

4.3. Captura de paquetes con PCAP

La biblioteca libpcap, resulta especialmente útil en este Trabajo de Fin de Grado, porque simplifica mucho el proceso de captura de paquetes. Gracias a dicha librería, se puede trabajar con paquetes en tiempo real desde la red o usando datos de un fichero .pcap.

Para capturar paquetes en tiempo real se hace uso de la función pcap_open_live().

Para la lectura de capturas anteriores desde un archivo de datos pcap, se hace uso de la función pcap_open_offline().

Para poder analizar individualmente cada paquete, se usa la función pcap_loop(). Esta función se encarga de enviar a la función analizar_paquete() la secuencia de bits que analizará el disector de paquetes.

Otra de las ventajas de la librería libpcap, es que permite el filtrado de paquetes de forma muy sencilla. Haciendo uso de las funciones pcap_compile() y pcap_setfilter(), se puede pasar por terminal el filtro que se desea. Lamentablemente, no se puede filtrar el protocolo QUIC, pero sí UDP. De esta forma evitamos que el tráfico TCP sea procesado.[14]

4.4. Arquitectura desarrollada

Para este disector de tráfico, se ha usado el lenguaje de programación C. Principalmente, por el conocimiento previo que se tenía sobre este y su alta velocidad de procesamiento. Pero también, por su facilidad para trabajar con direcciones de memoria.

Como se puede ver en la **Figura 4.1**. El disector cuenta de tres partes principales, main(), analizar_paquete() e insert().

La función main() se encarga del proceso de lectura de valores del comando en la terminal. Posteriormente, procede a iniciar el proceso de la librería pcap, para el filtrado de paquetes elegido en el comando. Por último, imprime los campos que se van a analizar en el disector y llama la función pcap_loop(). Como se ha comentado anteriormente, esta función envía el array de bits de cada paquete a la función analizar_paquete.

Esta función permite parsear un paquete individual y obtener su información, en la sección **4.4.3**, se explica con más detalle su funcionamiento.

La última función a destacar es insert(). Esta función permite añadir la información de cada paquete a un resumen de los datos de las sesión. En la sección **4.4.4** se explica detalladamente.

4.4.1. Precarga de memoria

El objetivo de la precarga de memoria es optimizar el rendimiento del disector, en el apartado **4.5 Optimización** se explicarán los motivos con más detalles. Lo primero que hace el programa es una petición al sistema para la reserva de una gran cantidad de memoria. De esta forma, cuando se reciba una nueva sesión, se puede empezar a escribir

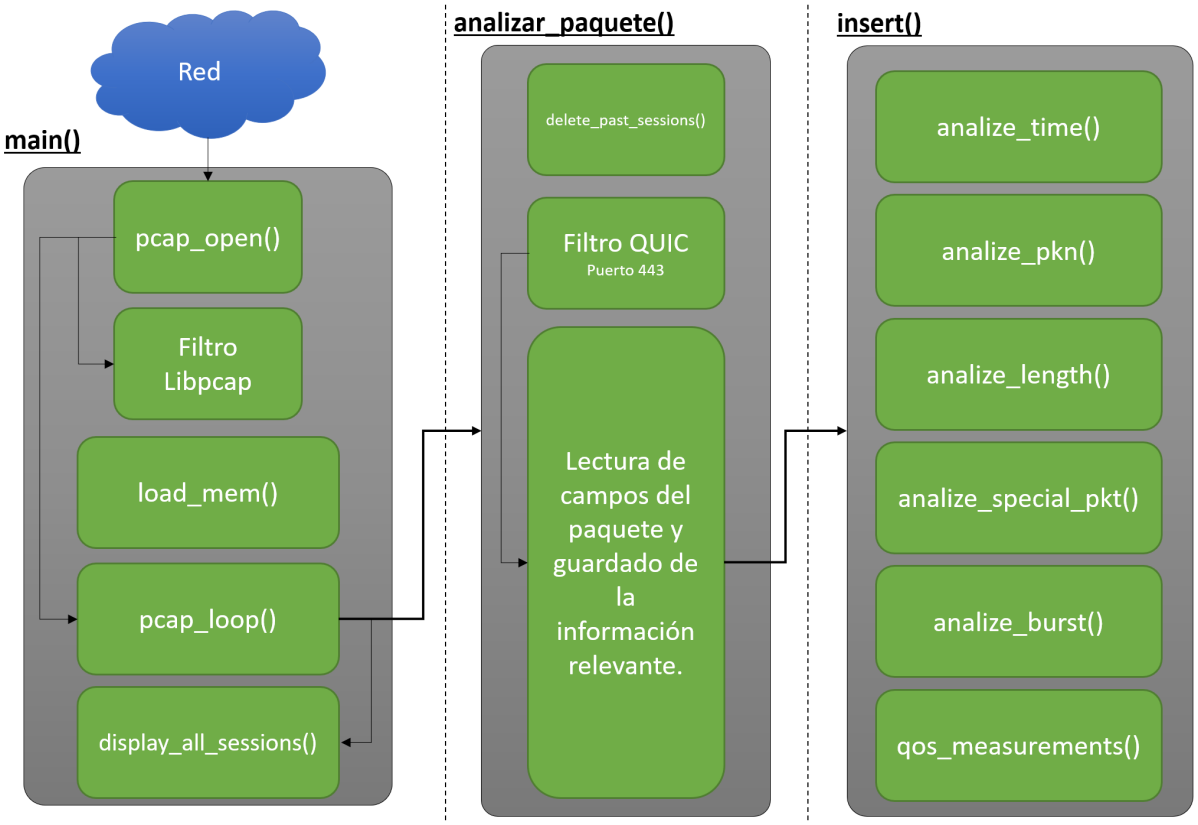

Figura 4.1: Esquema del código

en la memoria directamente. Si bien se podría realizar la reserva de memoria cada vez que hay una nueva sesión. Este sistema se muestra ineficiente cuando se trabaja en redes con mucho tráfico, debido a que cada reserva de memoria es una solicitud al sistema, que paraliza el disector y podría provocar la pérdida de paquetes. Con una precarga de memoria añadimos un tiempo de espera extra al comienzo del sistema, pero una vez terminada la reserva el disector hará muy pocas paradas por este motivo.

El disector se ha prefijado a 20000 sesiones simultáneas, en caso de querer añadir más sesiones o tener una memoria mayor o disminuir, este valor se puede cambiar, pero en tal caso también se recomienda ajustar el número de paquetes necesarios para la llamada a la rutina delete_past_sessions(). Más adelante, se comentará le uso que se le da.

4.4.2. Filtrado de paquetes

En este trabajo se busca la disección del protocolo QUIC únicamente. Por lo tanto, aquellos protocolos distintos no nos interesa analizarlos y cuanto antes sean filtrados, menos tiempo de análisis de paquetes útiles se pierde. Por ello, se han implementado dos filtros.

El primero sería usando la biblioteca libpcap, como se ha comentado anteriormente. Este filtro se puede invocar desde el terminal que se lanza el programa. Usando la opción -e. Por ejemplo:

Para quedarnos únicamente con paquetes UDP podríamos usar: *./disector -i eth0 -e udp*. Para ver una sesión en concreto podríamos usar: *./disector -i eth0 -e "src or dst*

127.216.43.1".

El segundo filtro estaría incorporado en la función analizar_paquete(). Su funcionamiento es comprobar el campo puerto origen o destino de la cabecera UDP. Si alguno de ellos no es 443. Entonces el protocolo no será QUIC y no se seguirá analizando. Este puerto puede ser cambiado mediante la opción -p.

Por ejemplo: *./disector -i eth0 -e udp -n 10001 -p 444*

El comando anterior filtraría por udp, buscaría tráfico QUIC en el puerto 444 y cada 10001 paquetes llevaría a cabo la rutina de borrado de sesiones caducadas.

4.4.3. Análisis de la trama

En esta sección se va a comentar cómo se procesa la trama de tráfico que se recibe y qué campos se guardan.

Para procesar la trama, se usan dos métodos distintos, *casting* y *parsing*. El *casting* permite obtener todos los campos de cabecera *Ethernet*, IP y UDP de forma automática. Esto se puede hacer porque las cabeceras de estos protocolos no varían y se puede crear una estructura estática previa.

El *parsing* es el método que se utiliza para analizar el protocolo QUIC. Consiste en avanzar byte a byte en la cabecera y tomar decisiones según los datos leídos. Esto hay que hacerlo así porque como se vió en **Figura 2.5**, los campos como CID, *Version* o *Message Authentication Hash* pueden no aparecer en un paquete, según los valores del campo *Public Flags* podremos saberlo. Este campo, también dirá qué tamaño tiene el campo *Packet Number*.

De los protocolos *Ethernet*, IP y UDP se guardan los datos de tiempo de llegada del paquete, la longitud de la trama, direcciones IP y puertos de origen y destino. De la cabecera pública se guardarán los datos de CID, PKN, máximo PKN posible y versión.

Después, se procede al análisis de los campos de la cabecera ACK y cabecera Special Packet. Los paquetes especiales con *Client Hello* y *Rejection* son especialmente importantes de analizar, ya que son los que contienen la información de establecimiento de conexión.

Uno de los principales problemas observados en el análisis del protocolo GQUIC, es que no hay ningún campo que avise de la llegada de los campos de ACK o *Special Packet*. Lo que se ha observado es que siempre que llega un paquete con dichas características, la longitud de la trama es muy grande (más de 1300 Bytes). Por ello, a veces el disector detecta como paquetes especiales algunos paquetes normales, cuyo contenido cifrado coincide con lo que podría ser una cabecera de un paquete especial. Para evitar que este error se propague, se han tomado medidas que se comentarán en el apartado de guardado de sesión posterior.

Los campos que se guardan de los paquetes especiales son: *Estimated RTT*, UAID, AEAD, ICSL y *hostname* (obtenido del SNI). También se guarda si el paquete es CHLO, REJ, ACK, NACK, *Connection Close* u otro tipo de paquete.

4.4.4. Guardado de la sesión

La información de cada paquete se envía a la función insert(). En esta función se busca la sesión en la tabla *hash* con listas enlazadas [15][16]. El funcionamiento de la tabla *hash* queda representado en la **Figura 4.2**. En caso de no encontrar la sesión se crea una nueva. Los campos guardados para cada sesión se comentarán en el apartado **4.4.6 Presentación de resultados**.

La función insert() se encarga de procesar los datos obtenidos en cada paquete y va actualizando los datos de las sesión. Para ello, hace uso de 4 funciones adicionales:

- **analize_time:** Esta función se encarga de actualizar los campos de llegada del primer paquete y último paquete. También detecta cuando una sesión ha caducado. Si un paquete tarda más de 30 segundos (o lo que indique el campo ICSL) en llegar desde el último o se envía un paquete *Connection Close*, entonces la sesión se dará por terminada y no se actualizará más. Esta función también actualiza el campo *hostname* y UAID cuando corresponda.

- **analize_pkn:** Esta función tiene la tarea de actualizar el campo first pkn y last pkn, para ambos sentidos de la conexión. También, será la función encargada de detectar pérdidas de paquetes. Para ello, hace uso de los huecos entre paquetes. Esto quiere decir que, si cuando llega un paquete, este no es el inmediatamente posterior al último recibido, entonces se ha producido uno o varios huecos en la transmisión. Este método no permite saber con exactitud qué cantidad de paquetes se están perdiendo, pero una cantidad alta de huecos es un indicador fiable de que la conexión está teniendo problemas por altas pérdidas.[17]

  Una particularidad de QUIC es que cuando un pkn llega al máximo que permite su tamaño de campo. Puede actualizarse a un tamaño mayor y seguir aumentando o volver a valer 0. Esto se ha tenido en cuenta para no contabilizar huecos erróneos. Esta peculiaridad da lugar a que sea posible que en una sesión el último paquete recibido sea menor que el número total de paquetes recibidos.

- **analize_length:** Esta función actualiza para ambas direcciones de cada sesión el número de Bytes recibidos. Se guardan los valores para los protocolos IP, UDP y QUIC. También se guardan los Bytes recibidos sin cabeceras. Para calcular este dato hace falta que cada paquete descuente el número de cabeceras QUIC que ha recorrido. Ya que el resto del contenido del paquete será Payload cifrado, es decir, la información enviada.

- **analize_special_pkt:** Esta función se encarga de actualizar los parámetros asociados a los campos especiales, esto incluye los campos de los paquetes ACK. Cuando llega el primer CHLO se actualizan los campos IRTT y AEAD. También se guarda la información del primer paquete de cada tipo.

  Como el primer paquete de cada sesión es un Client Hello, normalmente todos los datos generales de la sesión quedan guardados. Por lo tanto, aunque al analizar un paquete se reconozca un Client Hello que no corresponde, los datos erróneos obtenidos no se guardarán en la sesión.

- **analize_burst:** El protocolo QUIC tiene todos los datos cifrados por lo tanto no se puede ver los detalles de HTTP/2 que transporta [18]. Esto hace especialmente

complicado el cálculo de algunas métricas. Por ello, se usa el método de ráfagas de paquetes. Cuando se envía y recibe una ráfaga de paquetes, es decir, muchos paquetes seguidos. Se puede asumir que ha habido una comunicación entre el cliente y el servidor. Por lo tanto, se puede calcular el RTT de la conexión usando el primer paquete enviado y el último recibido de la ráfaga.[11]

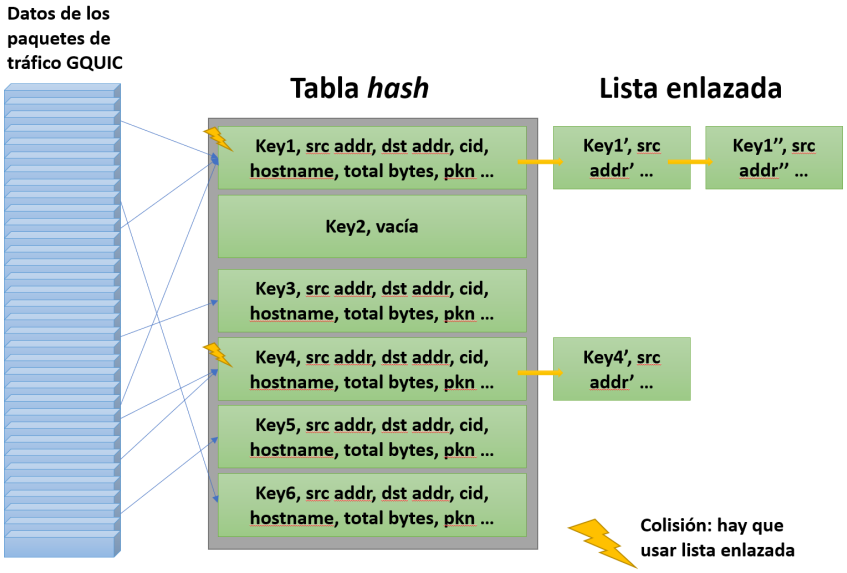

Figura 4.2: Estructura de la Tabla Hash

4.4.5. Limpiar memoria

El sistema de memoria del disector es una tabla *hash* con listas enlazadas para evitar las colisiones. Esto reduce significativamente la velocidad de búsqueda de una sesión, pues solo hay que calcular su posición en la tabla. Para calcular la posición en la tabla se usan los valores de las direcciones IP y de los puertos.

Para evitar que una sesión quede guardada indefinidamente en la memoria, hay que proceder a su borrado. A este proceso también se le conoce como: *Garbage collector*. En la función analizar_paquete() se lleva a cabo la llamada a esta rutina, se lanza cada 10000 paquetes de forma predeterminada, pero se puede cambiar con la opción -n en el terminal.

La función delete_past_sessions() se encarga de comprobar en todas las sesiones de la tabla *hash* cuándo ha llegado el último paquete de la sesión. Si este tiempo es mayor a un minuto del último paquete recibido entonces la sesión ha caducado y por lo tanto se imprime su información con la función print_session() y luego se reinician sus valores. De esta forma la sesión queda borrada de la memoria, pero puede ser reutilizada por otra sesión con la misma posición en la tabla *hash*.

4.4.6. Presentación de resultados

Cuando el disector termina de leer los datos de un archivo .pcap o se para la captura en una interfaz de red, se deben presentar los resultados de cada sesión. Para ello, se imprime

en una línea de texto todos los campos guardados para cada sesión. Estos campos impresos se pueden observar en **Figura 4.2** y son los siguientes:

Direcciones IP de origen y destino, puertos UDP de origen y destino, tiempo de llegada del primer y último paquete, CID y Hostname.

Para ambas direcciones de la conexión: Longitud de datos IP,UDP y QUIC. Con y sin cabeceras. Primer y último número de paquete, total de paquetes recibidos, total de huecos detectados.

También, se imprime la información del tiempo de llegada del primer paquete CHLO, REJ, ACK y NACK. Así como el total de paquetes con esas características recibidos.

Los datos más relevantes de la conexión, como pueden ser la versión, IRTT, UAID y AEAD. Así como si se ha recibido un *connection close* y cuándo.

El campo IRTT ofrece una estimación que hace el cliente del RTT. El campo UAID da el identificador de cliente. El campo AEAD ofrece el intercambio de algoritmos de cifrado que se va a usar.

Por último, se imprime la información para medidas de calidad de servicio, necesarias para saber si la conexión ha sido correcta. Éstas incluyen latencia, tasa de transferencia y pérdidas. La latencia, se calcula de dos formas: la primera con el método de ráfagas comentado anteriormente; la segunda, comparando los tiempos de llegada del primer paquete *Client Hello* y *Rejection*, ya que esto son los primeros paquetes que se envían en el establecimiento de conexión. La tasa de transferencia también tiene dos versiones, la primera es la tasa de transferencia total y la segunda es la tasa de transferencia de datos. La diferencia entre ambas reside en que en la segunda se tiene en cuenta las cabeceras de los paquetes.

4.5. Optimización

Uno de los objetivos de este disector es que sea capaz de trabajar en redes de alto rendimiento. Se puso el objetivo de 10 Gbit/segundo. Para lograr este objetivo, el disector debe gastar el mínimo tiempo posible en cada paquete. Esto se consigue mediante el descarte de los paquetes que pertenezcan a un protocolo distinto a QUIC. También, es importante que el guardado en memoria sea lo más rápido posible. Para lograr este objetivo, se optó por una tabla *hash*. Este tipo de estructura de datos permite que con el cálculo de un código *hash* se puede acceder directamente a la posición del array dónde se guarda la información de la sesión. Este sistema tiene un problema, y es que varias sesiones pueden coincidir sus códigos *hash*. Por ello, la tabla *hash* se complementa con un sistema de listas enlazadas.

Cuando se trabaja en redes con mucho tráfico aumenta la probabilidad de que varias sesiones coincidan sus códigos *hash* y creen listas enlazadas. Cada vez que un paquete de dichas sesiones se inserte, el disector deberá buscar la sesión en la lista enlazada, perdiendo tiempo y disminuyendo el rendimiento del disector. Por ello, se introdujo la función delete_past_sessions(). Las sesiones caducadas se imprimirán y dejarán su sitio en la tabla hash libre para las nuevas sesiones. Esta función es lenta porque busca en toda la lista las sesiones caducadas que pueden ser pocas o ninguna. Por ello, es importante que esta rutina se ejecute cada cierto tiempo. Hay que buscar un equilibrio entre el

número de sesiones que soporta el disector, que depende de la memoria que le podamos asignar y el número de veces que se lanza la rutina de borrado de sesiones caducadas. De forma predeterminada, se ejecuta la rutina cada 10000 paquetes recibidos, pero se puede cambiar. Si se disminuye ese número, puede ser que el sistema pase demasiado tiempo buscando sesiones caducadas. Mientras que si se aumenta el sistema empezará a tener sesiones caducadas que ocuparan espacio en la tabla hash.

La última optimización que se realizó es la precarga de memoria, cuyo funcionamiento se explicón en un apartado anterior. En vez de reservar memoria del sistema cada vez que aparece una nueva sesión, se prereserva una gran cantidad de memoria que luego se va cediendo a las nuevas sesiones que aparecen. De esta forma, el inicio del disector es más lento, pero a la larga el rendimiento es mayor, porque evitamos las reservas de memoria innecesarias. Este sistema no evita que cuando hay una colisión haya que reservar memoria para la nueva sesión en la lista enlazada. Pero es un coste mínimo e inevitable.

Con estas mejoras y el uso de la opción -O3 en la compilación, se consiguió que el disector trabajase a 7.5 Gbit/segundo.

4.6. Mejoras

Una vez conseguido que el disector sea funcional y dé resultados correctos, se llevaron a cabo tareas de mejora. Lo primero que se añadió fueron las medidas de calidad de servicio. Para analizar el rendimiento de cada sesión se añadieron los campos de latencia, tasa de transferencia y porcentaje de pérdidas. Estos campos se calculan una vez terminada la sesión, ya que para cálculo requieren de los valores finales de la sesión, como el número bytes en los paquetes o el tiempo que ha durado la sesión completa.

Para que el disector pueda ser ejecutado en equipos con capacidades distintas, se permite la configuración de algunos valores, en concreto resultará muy útil el número de paquetes necesarios para ejecutar la función delete_past_session().

Otra mejora es el sistema de alertas de fallos, cuando se introduce un filtro erróneo o no se puede capturar en una interfaz, el disector mostrará un mensaje de error y finalizará su ejecución. También se mostrarán mensajes de cómo invocar el comando correctamente, en caso de un error en la llamada al disector.

4.7. Conclusión

En este capítulo hemos detallado el proceso que se ha llevado a cabo para la elaboración del disector, teniendo en cuenta la teoría aprendida en el estudio del protocolo QUIC y teniendo como objetivos los requisitos propuestos. Se ha observado que el protocolo funcional GQUIC es ligeramente diferente al protocolo teórico QUIC. Y que aún no queda claro cómo se diferencian los paquetes especiales de paquetes con tráfico cifrado cuya información coincide con lo esperado en los campos de QUIC.

También se ha explicado qué optimizaciones y mejoras se han llevado a cabo. Una vez terminada la implementación del disector, se procederá a realizar las pruebas convenientes para comprobar su efectividad.

5 Resultados

5.1. Introducción

Una vez explicado el funcionamiento del disector se procede a comprobar su funcionamiento en entornos reales. Para ello, se han planteado tres situaciones distintas: búsqueda web de un periódico digital; visualización de un vídeo de 10 horas en Youtube y descarga de un archivo en Google Drive, añadiendo pérdidas de paquetes a esta última. Todas las capturas se han realizado desde la máquina virtual y se ha usado el navegador web Google Chrome. Estas trazas se han subido a github, junto al código.

5.2. Navegación web

Cuando se realiza una visita a una página web que no depende de los servidores de Google, es posible que no se capture tráfico QUIC, ya que dependerá del servidor que nos proporcione el servicio tenga implementado dicho protocolo, lo que no es común. Aún así, si se usa el navegador Google Chrome se espera capturar algo de tráfico QUIC, por la búsqueda que realiza el navegador y los anuncios que puedan tener las páginas web que visitamos. [6]

Esta traza se realizó buscando a través de Google Chrome un artículo en un periódico digital y más tarde se buscaron otros dos artículos más. Los resultados del disector nos permiten observar los campos obtenidos para cada sesión. También se muestran los resultados de QoS calculados una vez termina la conexión. Los resultados obtenidos se pueden ver en **Tabla A.3** y **Tabla A.4**. Se puede observar que aparecen 22 sesiones de QUIC. En la capa IP se han intercambiado 174.587 Bytes en la dirección cliente servidor y 2.492.850 Bytes en la dirección servidor a cliente. Contando solamente los datos intercambiados en QUIC, sin cabeceras de paquetes, 64.037 Bytes en la dirección cliente a servidor y 2.395.600 Bytes en la dirección servidor a cliente.

Los *packet number* que empiezan en 1 y el último recibido (en la tabla se llama pkn_max) coincide con el número total de paquetes recibidos, salvo alguna excepción. Hay algunas sesiones cuyos pkn_max y pkn_total no coinciden, esto ocurre cuando la sesión llega 255 y decide si continúa aumentando o trunca el valor a 0.

También se puede observar que en todas las sesiones la versión usada es Q043. Y no ha habido *Connection Close*, es decir, todas las sesiones han tenido cierre silencioso. Se puede observar que en la mayoría de sesiones los valores de IRTT, RTT burst y RTT calculated se aproximan bastante. Ha habido muy pocas pérdidas de paquetes, en todas las sesiones cercano al 0 % y el UAID y AEAD coinciden en todas las sesiones, lo que tiene sentido ya que todas las peticiones se han realizado desde la misma máquina virtual.

Por último, se puede apreciar que hay un valor en el campo time first NACK, al comprobar dicho paquete en Wireshark, se observa que es un error del disector. Dicho paquete no debería ser marcado como NACK.

5.3. Vídeo de Youtube

Esta traza se obtuvo generando tráfico mediante la visualización de un vídeo de Youtube de varias horas. El objetivo es observar el funcionamiento del protocolo en un entorno de Google, ya que Youtube pertenece a la empresa y tiene incorporado el protocolo GQUIC. El motivo por el que se capturó una traza de paquetes tan grande es para poder calcular la velocidad de procesamiento del disector. Como se observa en la **Tabla 5.1**. En total, la traza contiene 1,464 GBytes de tráfico GQUIC versión Q043.

Se han generado 1.152 conexiones. Se han enviado 59.960.460 Bytes del cliente al servidor en la capa IP. Se han recibido 1.351.344.504 Bytes. En la capa QUIC sin contar cabeceras, se han enviado 34.403.866 Bytes y se han recibido 1.316.596.031 Bytes.

Tabla 5.1: Resultados vídeo en Youtube

Num. Sesiones	1152
ctos_ip_len_total	59.960.460
stoc_ip_len_total	1.351.344.504
ctos_quic_header_len_total	34.403.866
stoc_quic_header_len_total	1.316.596.031

En esta prueba se puede observar el funcionamiento del RTT0. En la **Tabla 5.2** se observa que en algunas sesiones no aparece el tiempo de envío de *rejection*, que es el paquete que contiene la respuesta del servidor en el establecimiento de conexión. Esto ocurre porque anteriormente se ha realizado el establecimiento con dicho servidor. Se puede comprobar viendo que solo una sesión tiene el campo tiempo de envío de *rejection*.

Los resultados completos de esta captura no se han podido mostrar en la memoria por ocupar demasiado, pero se han subido a la página *Github: https://github.com/victor3k/DisectorGQUIC* para su consulta.

Tabla 5.2: Resultados Youtube para observar el funcionamiento de RTT0

1src_addr	2dst_addr	3src_port	4dst_port	5time_first_pkt	6time_last_pkt	7cid	8hostname	29time_first_chlo	30time_first_rej
192.168.241.182	216.58.211.35	32902	443	1563858908.278189	1563858921.697585	11026747837960699776	fonts.gstatic.com	1563858908.278189	1563858908.302024
192.168.241.182	216.58.211.35	37983	443	1563858915.522326	1563858948.459254	3214019597397302128	www.google.es	1563858915.522326	1563858915.546495
192.168.241.182	216.58.211.35	54335	443	1563859316.525436	1563859331.584676	10649574460776060103	beacons.gcp.gvt2.com	1563859316.525436	-1.000000
192.168.241.182	216.58.211.35	44625	443	1563859366.212761	1563859425.912128	11743958619630881926	beacons.gcp.gvt2.com	1563859366.212761	-1.000000
192.168.241.182	216.58.211.35	40795	443	1563859465.552494	1563859486.353435	10564698915791142354	beacons.gcp.gvt2.com	1563859465.552494	-1.000000
192.168.241.183	216.58.211.35	34189	443	1563859547.152832	1563859562.211715	42630736994652852886	beacons.gcp.gvt2.com	1563859547.152832	-1.000000
192.168.241.183	216.58.211.35	39022	443	1563859621.514432	1563859667.838759	63890726488860641150	beacons.gcp.gvt2.com	1563859621.514432	-1.000000
192.168.241.183	216.58.211.35	41197	443	1563859736.527893	1563859768.924627	4072460151889424747	beacons.gcp.gvt2.com	1563859736.527893	-1.000000
192.168.241.183	216.58.211.35	58205	443	1563860646.293171	1563860701.475656	12969751987637121101	beacons.gcp.gvt2.com	1563860646.293171	-1.000000
192.168.241.183	216.58.211.35	35306	443	1563860747.154000	1563860762.211338	9588284219688285770	beacons.gcp.gvt2.com	1563860747.154000	-1.000000
192.168.241.183	216.58.211.35	44683	443	1563860807.150190	1563860837.098427	17883986176134585982	beacons.gcp.gvt2.com	1563860807.150190	-1.000000
192.168.241.183	216.58.211.35	42875	443	1563860886.523925	1563860906.292921	12261652516098455261	beacons.gcp.gvt2.com	1563860886.523925	-1.000000
192.168.241.183	216.58.211.35	35645	443	1563861307.231416	1563861322.261482	13484299084134542819	beacons3.gvt2.com	1563861307.231416	-1.000000
192.168.241.183	216.58.211.35	59414	443	1563861427.237892	1563861442.270929	12144286861864097278	beacons3.gvt2.com	1563861427.237892	-1.000000
192.168.241.183	216.58.211.35	52358	443	1563862592.256338	1563862592.353710	14200450030094937364	beacons.gvt2.com	1563862592.256338	-1.000000
192.168.241.183	216.58.211.35	56899	443	1563863880.157337	1563863895.212435	15804055310915613442	beacons.gcp.gvt2.com	1563863880.157337	-1.000000
192.168.241.183	216.58.211.35	49105	443	1563863963.738609	1563863978.794427	7065743971225248469	beacons.gcp.gvt2.com	1563863963.738609	-1.000000
192.168.241.183	216.58.211.35	37267	443	1563865818.058040	1563865833.118243	15158114797273470390	beacons5.gvt3.com	1563865818.058040	-1.000000
192.168.241.183	216.58.211.35	41824	443	1563866600.423941	1563866615.454756	5547902267318584804	beacons3.gvt2.com	1563866600.423941	-1.000000

5.4. Descarga de Google Drive con pérdidas

Finamente, se añadió la opción de VMWare para añadir pérdidas en los paquetes de la red virtual. Se pusieron 5 % de pérdidas de paquetes en la descarga. Si se ponen más pérdidas el servidor no transmite con QUIC, sino que empieza a transmitir por TCP + TLS. [19]

Después de añadir las pérdidas se procedió a descargar una carpeta de imágenes desde Google Drive, se ha usado el navegador Google Chrome.

En la **Tabla 5.3** se puede observar que, se han generado 50 conexiones de QUIC, en esta captura se puede observar que los resultados de porcentaje de pérdidas son mayores que en los anteriores resultados. La media de pérdidas es de 5.25 % en la descarga aproximadamente, si este valor no coincide con la pérdidas de la máquina virtual es porque el cálculo que hace el disector es sobre los huecos que quedan entre los números de paquetes. Por ejemplo, si se pierde un paquete, el disector contará cuantos huecos hay entre el paquete recibido y el esperado. En la carga de paquetes, es decir, dirección cliente a servidor, las pérdidas son mucho menores, como no se ha añadido pérdidas con VMWare, solo cuenta las de la red.

Por último, a mayor número de pérdidas cabría esperar que aumentase el número de asentimientos. Pero se puede observar que el número de ACK y NACK obtenidos no ha aumentado con respecto a los resultados anteriores. Esto puede deberse a que una vez establecida la conexión los asentimientos se realizan junto con los datos cifrados. [20]

Al igual que los resultados del vídeo en Youtube, los resultados completos de esta captura no se han podido mostrar en la memoria por ocupar demasiado, pero se han subido a la página
Github: https://github.com/victor3k/DisectorGQUIC para su consulta.

5.5. Comparación entre los resultados

Para poder comprobar si el desarrollo de la optimización del disector ha sido útil, se han llevado a cabo tests en los tres escenarios planteados anteriormente, comparando los resultados de tiempo del disector con y sin la optimización activada. Estos resultados

Tabla 5.3: Resultados descarga de un archivo en Google Drive con 5 % de pérdidas de paquetes

Num. Sesiones	50
ctos_ip_len_total	5.446.103
stoc_ip_len_total	47.845.776
ctos_quic_header_len_total	4.343.903
stoc_quic_header_len_total	46.643.424
ctos_loss	0,93%
stoc_loss	5,25%
Num ACK	31
Num NACK	0

quedan mostrados en la **Figura 5.1**. Para este test, se han usado 100 sesiones en memoria y 10000 paquetes necesarios para ejecutar la rutina de borrado de sesiones. Para desactivar la optimización, se hace uso de la opción -n, con esta opción podemos cambiar el número de paquetes necesarios para que se lleve a cabo la rutina de delete_past_sessions(). Si se pone un número de paquetes muy elevado, nunca se ejecutará.

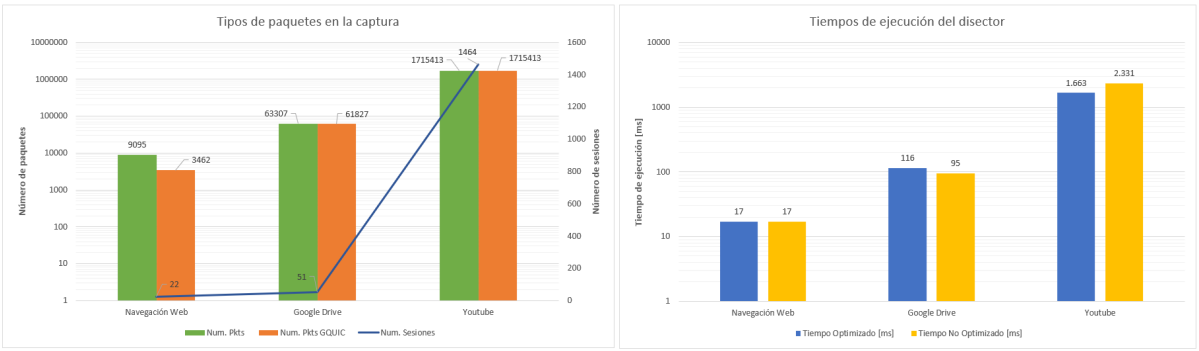

Figura 5.1: Comparación de los resultados obtenidos en rendimiento

En el gráfico de la derecha podemos observar algunas diferencias entre los resultados: Para la captura del periódico, la optimización no afecta en el rendimiento; para la captura de la descarga mediante Google Drive, la optimización disminuye el rendimiento y por último la captura de vídeo en Youtube se aprecia una mejora de rendimiento con la optimización.

Para explicar este comportamiento, es necesario revisar los datos del gráfico de la izquierda, en dicho gráfico se muestran las características de la captura en número de paquetes, número de paquetes de protocolo GQUIC y número de sesiones obtenidas. La captura de tráfico generado por la visita web, tiene muy pocos paquetes GQUIC, 3462. Son menos de la mitad de los paquetes necesarios para que se ejecute la rutina de borrado de sesiones caducadas, por ese motivo, no se aprecia ningún cambio.

En la captura de paquetes por la descarga de Google Drive, sí se aprecia un descenso en el rendimiento, esto se explica porque hay 61827 paquetes GQUIC, es decir, se ejecuta 6 veces la rutina de borrado de sesiones, pero con solo 51 sesiones es improbable que haya habido colisiones que pudieran disminuir significativamente el rendimiento del disector.

Por último, podemos observar que en el caso del vídeo de Youtube sí se ha disminuido

el tiempo con la optimización. Esto ocurre porque en la versión sin optimización, es seguro que las 1464 sesiones han creado listas enlazadas, las búsquedas en dichas listas provocan que el rendimiento disminuya significativamente. En la versión optimizada al haber un número elevado de paquetes, se asegura que el disector borra las sesiones caducadas de forma frecuente, evitando el problema de las colisiones entre sesiones y por ello, aumenta el rendimiento.

5.6. Sesión modificada con SCAPY

Uno de los problemas que tiene UDP con respecto a TCP es que no hay ningún campo que indique qué protocolo lleva encapsulado. Por ello se hace uso de los puertos bien conocidos, por ejemplo, DNS siempre usará el puerto 53. En cambio, el puerto de QUIC, 443 queda definido como el puerto bien conocido para HTTP en UDP, esto plantea la siguiente duda. Qué resultado da el disector en caso de que un protocolo distinto a QUIC se transmita por el puerto 443. Como se puede observar en la **Figura 5.2**, para llevar a cabo este experimento se ha tomado una captura de tráfico UDP distinto a QUIC y con ayuda del programa SCAPY se han cambiado los puertos a 443. En la **Tabla 5.4** se puede observar los resultados del disector a la sesión modificada anteriormente.

No.	Time	Source	Destination	SRC port	DST port
1	0.000000	192.168.241.1	255.255.255.255	17500	17500
2	0.008692	192.168.241.1	255.255.255.255	17500	17500
3	0.017351	192.168.241.1	255.255.255.255	17500	17500
4	0.017378	192.168.241.1	192.168.241.255	17500	17500
5	0.020332	192.168.241.1	255.255.255.255	17500	17500
6	0.020601	192.168.241.1	255.255.255.255	17500	17500
7	11.771132	192.168.241.216	192.168.241.2	51478	53
8	11.771661	192.168.241.216	192.168.241.2	36401	53
9	11.782482	192.168.241.2	192.168.241.216	53	51478
10	11.782522	192.168.241.2	192.168.241.216	53	36401
11	12.602925	192.168.241.216	192.168.241.2	43352	53
12	12.620482	192.168.241.216	192.168.241.2	40178	53
13	12.646109	192.168.241.2	192.168.241.216	53	43352
14	12.664687	192.168.241.2	192.168.241.216	53	40178
15	16.106135	192.168.241.216	192.168.241.2	57167	53
16	16.106576	192.168.241.216	192.168.241.2	42124	53
17	16.405712	192.168.241.216	192.168.241.2	47887	53
18	16.406056	192.168.241.216	192.168.241.2	49084	53

No.	Time	Source	Destination	SRC port	DST port
1	0.000000	192.168.241.1	255.255.255.255	443	443
2	0.008692	192.168.241.1	255.255.255.255	443	443
3	0.017351	192.168.241.1	255.255.255.255	443	443
4	0.017378	192.168.241.1	192.168.241.255	443	443
5	0.020332	192.168.241.1	255.255.255.255	443	443
6	0.020601	192.168.241.1	255.255.255.255	443	443
7	11.771132	192.168.241.216	192.168.241.2	443	443
8	11.771661	192.168.241.216	192.168.241.2	443	443
9	11.782482	192.168.241.2	192.168.241.216	443	443
10	11.782522	192.168.241.2	192.168.241.216	443	443
11	12.602925	192.168.241.216	192.168.241.2	443	443
12	12.620482	192.168.241.216	192.168.241.2	443	443
13	12.646109	192.168.241.2	192.168.241.216	443	443
14	12.664687	192.168.241.2	192.168.241.216	443	443
15	16.106135	192.168.241.216	192.168.241.2	443	443
16	16.106576	192.168.241.216	192.168.241.2	443	443
17	16.405712	192.168.241.216	192.168.241.2	443	443
18	16.406056	192.168.241.216	192.168.241.2	443	443

Figura 5.2: Paquetes UDP con puertos cambiados a 443 con SCAPY

En teoría, el disector analizará el tráfico, supuestamente ilegible como si fuesen campos del protocolo QUIC. Algunas de las anomalías que deberían aparecer son: Que los números de paquete no empiecen en 1 y no sigan ningún orden lógico, esto hará que haya muchos huecos. La versión del protocolo será NULL o distinta de Q043; No deberían aparecer los campos CHLO, REJ, ACK o NACK, por lo tanto, tampoco aparecerán los campos Hostname, IRTT, AEAD y UAID.

En la **Tabla 5.4** podemos comprobar que las predicciones son correctas. Cuando un protocolo que no es GQUIC es analizado por el disector, genera sesiones con valores atípicos.

Para minimizar el número de paquetes que se analizan de protocolos distintos a GQUIC. Se usa el valore reserved del campo public flags, este valor debería ser siempre 0, por ello, cuando hay un paquete con dicho valor a 1 se procede a descartar el paquete. Aún así, puede ocurrir que haya paquetes de otros protocolos que tengan un 0

en el valor de reserved, por lo tanto, se analizarían. La mejor forma de saber si un protocolo no es GQUIC Q043 es comprobar el valor del campo Version, este campo pondrá Q043 en el disector siempre que encuentre dicho valor. En caso de no encontrar el campo Version, pondrá NULL y en caso de que sea una versión de QUIC distinta a la estudiada, también pondrá NULL. Esto quiere decir que en caso de encontrar NULL en el campo Version en el resultado del disector, conviene comprobar el resto de campos de la sesión y si se encuentran más anomalías como las comentadas anteriormente se debe proceder al descarte de dicha sesión.

Una sesión con pérdidas muy altas también puede indicar que es un protocolo incorrecto. Esto se debe a que el protocolo QUIC manda los números de secuencia de manera ordenada, por ello, en caso de analizar los valores correspondientes al *packet number* de un protocolo distinto a QUIC se observarán valores aleatorios. En la **Tabla 5.4** se observa que una de las sesiones tiene un 50 % de huecos.

Tabla 5.4: Resultados sesión modificada con SCAPY

4dst_port	8hostname	21ctos_first_pkn	23ctos_pkn_total	24ctos_void	29time_first_chlo	30time_first_rej	31time_first_ack	32time_first_nack	37version	39irtt	46uaid	47aead
443	NULL	1563172898	22	2	-1.000000	-1.000000	-1.000000	-1.000000	NULL	-1	NULL	NULL
443	NULL	1684370533	105	58	-1.000000	-1.000000	-1.000000	-1.000000	NULL	-1	NULL	NULL
443	NULL	1563172898	6	2	-1.000000	-1.000000	-1.000000	-1.000000	NULL	-1	NULL	NULL
443	NULL	84	4	1	-1.000000	-1.000000	-1.000000	-1.000000	NULL	-1	NULL	NULL

5.7. Conclusión

En esta sección, se han comprobado los resultados del disector en tres escenarios distintos. Se ha podido verificar que el disector obtiene correctamente la mayoría de los campos, aunque a veces falla en la detección de paquetes ACK y NACK. Si bien la información de las conexiones no queda afectada por ello. También se ha comparado el rendimiento en los tres escenarios y se ha analizado el impacto que tiene la optimización, llevada a cabo al rendimiento del disector. Por último, se ha comprobado que resultado da el disector en caso de recibir un protocolo distinto a QUIC por el puerto 443 de UDP.

A continuación, se deberá proceder a contrastar los resultados del disector, comparándolos directamente con los resultados obtenidos por el disector de Wireshark.

6 Validación

6.1. Introducción

En este capítulo se explicará las comprobaciones que se han llevado a cabo para validar si los resultados del disector son similares a los obtenidos por el disector Wireshark. En principio, deberían parecerse bastante, ya que dicha herramienta es la que se usó durante el desarrollo para comprobar el funcionamiento del protocolo GQUIC.

Para llevar a cabo la validación se han planteado dos aproximaciones: la primera, consiste en una revisión manual de los datos obtenidos, usando las herramientas de Excel; después, se ha implementado un script con AWK para comparar los campos de varias sesiones a la vez.

6.2. Script Tshark

Tshark es una versión de Wireshark que funciona desde un terminal. Como analizador de protocolos, puede capturar tráfico desde la red o leer desde archivos guardados con anterioridad. Se optará por la segunda versión, de esta forma se podrá utilizar la misma captura en el disector y en tshark.

El funcionamiento del comando que invocamos es bastante simple, primero se establece el archivo del que se quiere leer los datos, luego un filtro de captura para desprenderse del tráfico no deseado. Por último, se establecen los campos se desean leer.[21]

Una vez termina de procesar el archivo nos guardará los campos de cada paquete en una línea en un archivo txt.

6.3. Comparación Excel

La primera verificación de funcionamiento del disector se llevó a cabo en Excel, una herramienta de cálculo desarrollada por Microsoft. Haciendo uso de la función importar datos desde archivo de texto, podemos añadir a la hoja de cálculo los datos obtenidos por el script de tshark.

Para comparar los datos obtenidos por el disector y los de tshark en las hojas de cálculo se han usado las funciones de filtrado para obtener los datos de una única sesión. Luego, se utilizaron las funciones de SUMA para contar los bytes en las cabeceras.

Como se puede observar en la **Tabla A.5** casi todos los campos son correctos. Los campos relacionados con la longitud de datos en GQUIC, es decir, descontando las cabeceras IP, UDP y GQUIC no se pueden validar correctamente. También se pudo observar que el campo AEAD no se mostraba correctamente, el disector daba los resultados en hexadecimal, cuando el disector Wireshark los muestra en ASCII. Esto se pudo corregir, pero requirió cambiar cómo se captura, guarda e imprime dicho campo.

Esta validación nos ha permitido comprobar que a rasgos generales el disector funciona bien, pero, hay que comprobarlo para sesiones más grandes y para tráfico más complejo. Para dicha tarea el análisis manual resulta inviable.

6.4. Comparación con AWK

Para llevar a cabo la validación de varias sesiones al mismo tiempo y sesiones con más tráfico, se ha decidido utilizar un programa en AWK. AWK es un lenguaje diseñado para procesar datos basados en texto. El objetivo es que a partir de la información por paquete proporcionada por Tshark se pase a datos de sesión usando las fórmulas de AWK. Además, se debe repetir este proceso para todas las sesiones disponibles.

Haciendo uso de los *associative arrays* que implementa AWK, se pueden obtener datos de cada sesión leyendo en cada línea del texto y comprobando que corresponda a la sesión buscada, después se puede acceder a la columna donde se encuentra el campo buscado. Dicho campo se puede guardar en el array asociativo o ser usado para operaciones posteriores.

Se pueden validar hasta 43 campos de los 47 que muestra el disector. Los 4 campos que no se pueden validar son aquellos referidos a la longitud de cabecera QUIC, que no se pueden obtener utilizando Tshark. Tampoco se valida RTTburst debido a la complejidad que representa. En la **Figura 6.1** se puede observar el resultado del *script*.

Para un archivo de 12 sesiones GQUIC se comprueban 516 campos, de los cuales 469 se obtienen como correctos, como se observa en la **Figura 6.3 (1)**. Es decir, un 90.89 % de los campos del disector coinciden con los obtenidos por Tshark. Al comprobar el 10 % que no coincide se pudo observar distintos tipos de fallos.

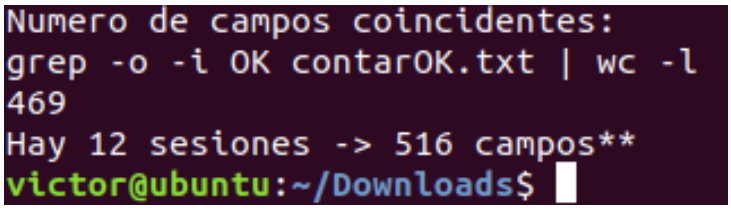

Figura 6.1: Resultados del script AWK.

- **Precisión de datos:** Los resultados obtenidos por el disector, no mostraban el mismo número de decimales que los resultados obtenidos por tshark, por ese motivo, la validación los detectaba como incorrectos. Ejemplo de esto es el campo porcentaje de pérdidas que se observa en la **Figura 6.3 (2)**.

- **Interpretación de caracteres:** Este fallo se debe a que AWK interpreta los espacios como separadores de campo. Pero algunos campos, concretamente el UAID da como resultado un string con varios valores separados por espacios. Por ello, AWK interpreta como campo UAID únicamente la primera parte del string, obviando el resto de los valores y no los compara correctamente con los obtenidos por el disector. Se puede observar en la **Figura 6.3 (3)**.

- **Campo AEAD:** Como se observa en la **Figura 6.3 (4)**. El campo AEAD no coincide siempre, porque cuando se envía en un CHLO, se envían unos valores y cuando responde un REJ, se devuelven otros valores. Por ese motivo, el disector implementado solo captura los campos enviados en el CHLO como válidos, el resto los omite. Investigando el motivo del fallo en la captura, y como se observa en la **Figura 6.2**, en los paquetes REJ, el campo AEAD pertenece al campo SCFG (*Server Configuration*). Por lo tanto, el disector debería buscar el campo SCFG y capturar sus valores en bytes, dentro de esos bytes, buscar el campo AEAD.

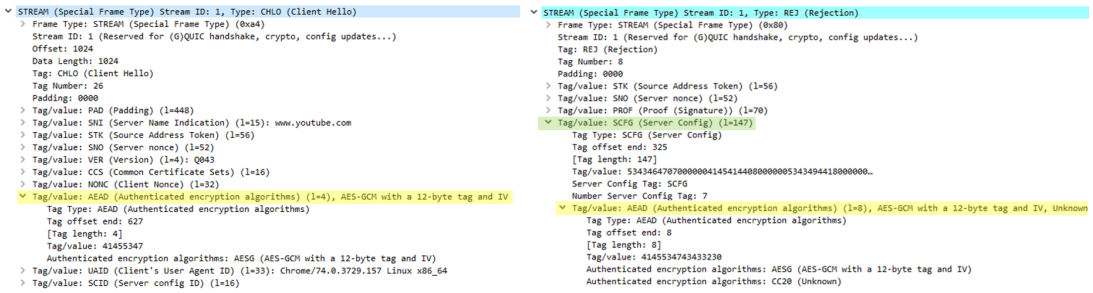

Figura 6.2: Campo AEAD y SCFG en paquetes CHLO y REJ.

- **ACK y NACK:** Por último, hay diferencias entre las veces que el disector detecta ACK y NACK y las veces que Wireshark los detecta. Este fallo ya se comentó anteriormente, en el apartado de Desarrollo. Se debe a que a veces los datos cifrados del Payload coinciden con los valores esperados por el disector para detectar ACKs y NACKs. Estos valores no tienen un aviso previo. Por lo tanto, el disector detecta más veces dichos paquetes que las que detecta Wireshark.

```
0.000000    |   0 (1)
OK
0.025240    |   0.0252399(2)
Chrome/74.0.3729.157    Linux   x86_64(3)
AESG    |   AESG,CC20 (4)
192.168.241.144 192.168.241.144
OK
172.217.168.164 172.217.168.164
OK
```

Figura 6.3: Errores detectados en AWK.

Por último, cabe mencionar que la validación por AWK es útil únicamente para trazas de tamaño pequeño o mediano. Como se puede observar en **Figura 6.4** para trazas grandes tiene tiempos de ejecución muy largos. La validación de la traza con el vídeo de Youtube largo tardó varios minutos en procesar.

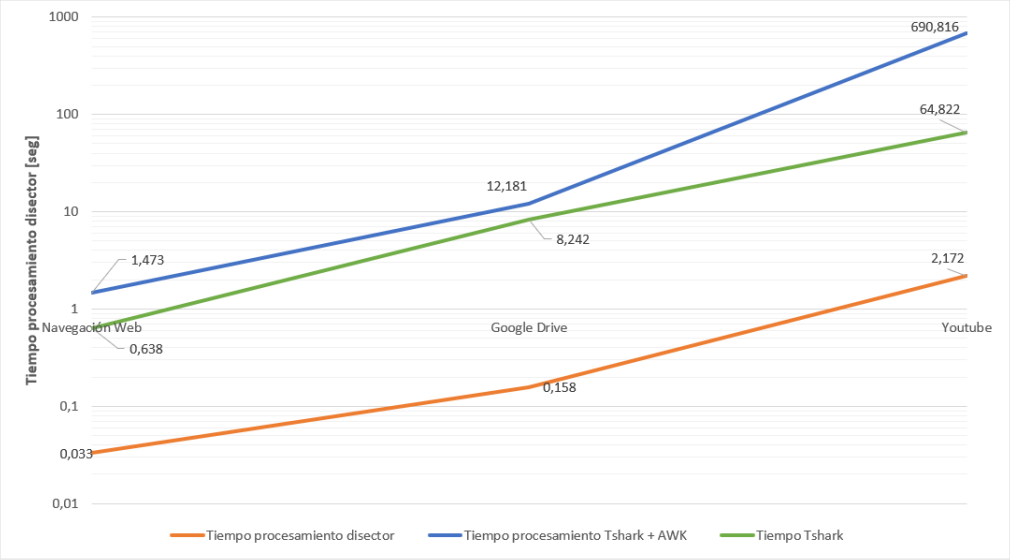

Figura 6.4: Comparación entre los tiempos de procesamiento del disector y Tshark + AWK.

6.5. Conclusión

En este capítulo se ha puesto a prueba los resultados obtenidos por el disector, en comparación con un disector avanzado como es Wireshark. Los resultados obtenidos son bastante positivos ya que los únicos fallos detectados han podido ser detectados y solucionados. El único error que persiste es el de los campos ACK y NACK.

Estos resultados nos permiten comprobar la utilidad del disector ya que obtiene resultados similares a los obtenidos por Wireshark además de añadir los valores de calidad de servicio y los datos obtenidos descontando las cabeceras.

También se ha comprobado las ventajas en tiempos de ejecución de usar un código en C frente a programas externos como tshark y AWK.

7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este Trabajo de Fin de Grado se ha llevado a cabo un estudio del protocolo QUIC. Primero, se ha comparado con los protocolos clásicos de Internet, TCP y UDP. Luego se ha llevado a cabo un estudio de las características del protocolo y su funcionamiento. Después se ha estudiado la implementación de dicho protocolo en Google Chrome, el navegador web de Google. Se ha estudiado la estructura de campos de la cabecera, para posteriormente implementar un disector capaz de extraer de cada paquete la información relevante. Estos datos, se procesan por sesiones y se muestran al usuario.

Para probar el funcionamiento del protocolo, se han llevado a cabo tres escenarios de pruebas diferentes y posteriormente se ha procedido a validar los resultados del disector con los obtenidos por la herramienta Tshark. De esta manera, se ha podido comprobar el correcto funcionamiento del mismo.

Tras la validación de los resultados, se puede concluir que el disector tiene un funcionamiento satisfactorio. Los campos mostrados se muestran coherentes con los obtenidos por la herramienta Wireshark. Su rendimiento es cercano a los 10 Gbit/s. Y muestra resultados extra de calidad de servicio.

La implementación del disector se ha llevado a cabo en C por las ventajas que ofrece [22]. Entre ellas, se encuentra el uso de la librería LibPCAP y el alto rendimiento que se puede alcanzar frente a otros lenguajes de programación. El diseño del disector se ha basado en el procesador de paquetes que se usan en las prácticas de redes de computadoras. El código está disponible en la página web:

Se puede consultar el código del disector y los resultados obtenidos en la pÃągina *Github: https://github.com/victor3k/DisectorGQUIC.*

Para el desarrollo del proyecto, se han requerido los conocimientos de varias asignaturas, como son Programación I y II, Redes de Comunicaciones I y II, Redes Multimedia y Sistemas Distribuidos. En consecuencia, a la realización del trabajo se han aumentado

los conocimientos en las asignaturas de Redes. Esto se debe al estudio de los protocolos TCP y UDP y al desarrollo del protocolo QUIC para satisfacer la demanda actual en contenido multimedia.

7.2. Trabajo futuro

El desarrollo del disector es, a mi parecer, bastante completo. Los campos que no se muestran al usuario son aquellos que dependen de números aleatorios como llaves de algoritmos de cifrado o claves de autenticación. Se podría, eso sí, completar en el estudio de distintos tipos de paquetes que no se han observado. Para poder llevar a cabo dicha tarea, se requeriría de una captura de red más completa, con varios clientes y servidores. Esto se intentó llevar a cabo en la red de la universidad, pero no se pudo realizar finalmente.

En concreto, se deberían poder observar paquetes con los campos BLOCKED, WINDOWS_UPDATE, GOAWAY, PING y STOP_WAITING. Otra mejora aplicable al disector es la elección de distintas versiones ya que en este desarrollo solo se ha llevado a cabo para la Q043 que es la más actualizada que mostraba Google Chrome.

Por último, en estudios del mismo departamento con el que se ha realizado este trabajo, se ha estudiado la detectabilidad de amenazas e intrusiones en las redes usando estadística. Igual, sería posible llevar a cabo un estudio al respecto para el protocolo QUIC.

Finalmente, se podría mejorar la gestión de la memoria y el uso de las tablas *hash*, ya que actualmente es funcional, pero no tiene un funcionamiento óptimo. Se podría sustituir la rutina de borrado de sesiones antiguas por una lista doblemente enlazada. De forma, que comprobando la sesión más antigua se podría ver si es necesario borrar dicha sesión de la memoria o no, en vez de recorrer todas las sesiones existentes.

Bibliografía

[1] S. Ohtsu, "Technical Overview of QUIC," https://es.slideshare.net/shigeki_ohtsu/quic-overview, Noviembre 2014, Último acceso: 14/10/2019.

[2] F. Lardinois, "Google wants to speed up the web with its quic protocol," https://techcrunch.com/2015/04/18/google-wants-to-speed-up-the-web-with-its-quic-protocol/, Abril 2015, Último acceso: 14/10/2019.

[3] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with quic: Design approaches and research challenges," *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, Marzo 2017.

[4] A. Saverimoutou, B. Mathieu, and S. Vaton, "Influence of internet protocols and cdn on web browsing," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Islas Canarias, España, June 2019, pp. 1–5.

[5] J. Chadwick, "Internet Encryption Hits 50 %: Netflix Eating 15 % of Global Traffic," https://www.cbronline.com/news/internet-encryption-sandvine, Octubre 2018, Último acceso: 14/10/2019.

[6] M. Trevisan, D. Giordano, I. Drago, M. Mellia, and M. Munafo, "Five years at the edge: watching internet from the isp network," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. Heraklion, Grecia: ACM, 2018, pp. 1–12.

[7] C. Yu, "Gquic protocol analyzer for zeek (bro) network security monitor," https://github.com/salesforce/GQUIC_Protocol_Analyzer, Agosto 2019, Último acceso: 14/10/2019.

[8] E. J. Iyengar and E. M. Thomson, "Quic: A udp-based multiplexed and secure transport draft-ietf-quic-transport-22," https://tools.ietf.org/html/draft-ietf-quic-transport-22, Julio 2019, Último acceso: 14/10/2019.

[9] U. C. Shah, "Flow-based analysis of quic protocol," Masaryk University, República Checa, Otoño 2018.

[10] J. Iyengar, "Quic loss detection and congestion control," https://www.tcpdump.org/pcap.html, Octubre 2019, Último acceso: 14/10/2019.

[11] C. López, D. Morato, E. Magaña, and M. Izal, "Effective analysis of secure web response time," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*. Paris, Francia: IEEE, Junio 2019, pp. 145–152.

[12] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in *IEEE Globecom 2006*. San Francisco, EEUU: IEEE, Enero 2006, pp. 1–5.

[13] F. Michel, Q. De Coninck, and O. Bonaventure, "Adding forward erasure correction to quic," *arXiv preprint arXiv:1809.04822*, Septiembre 2018.

[14] T. Carstens, "Programming with pcap," https://github.com/quicwg/base-drafts/blob/master/draft-ietf-quic-recovery.md, Septiembre 2019, Último acceso: 14/10/2019.

[15] F. Pfenning, "Principles of imperative computation. hash tables," https://www.cs.cmu.edu/~fp/courses/15122-f10/lectures/22-mem/hashtable.c, pp. 15–122, Noviembre 2010, Último acceso: 14/10/2019.

[16] M. Montilla Carrascosa, "Desarrollo de un sistema de monitorización de servicios de vídeo sobre redes IP con señalización RTSP," Trabajo de Fin de Grado, Grado en Ingeniería de Tecnologías y Sistemas de Telecomunicación. Universidad Autónoma de Madrid, Julio 2018.

[17] E. Miravalls Sierra, "Automatización y detección de anomalías en tráfico de Internet," Trabajo de Fin de Grado, Doble Grado en Ingeniería Informática y Matemáticas. Universidad Autónoma de Madrid, Junio 2016.

[18] P. De Vaere, T. Bühler, M. Kühlewind, and B. Trammell, "Three bits suffice: Explicit support for passive measurement of internet latency in quic and tcp," in *Proceedings of the Internet Measurement Conference 2018*. Boston, EEUU: ACM, Noviembre 2018, pp. 22–28.

[19] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is quic? provable security and performance analyses," in *2015 IEEE Symposium on Security and Privacy*. San Jose, EEUU: IEEE, Julio 2015, pp. 214–231.

[20] J. Roskind, "QUIC Design Document and Specification. Multiplexed Stream Transport over UDP," https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit#, Julio 2013, Último acceso: 14/10/2019.

[21] A. Walding, "Using Wireshark to Analyze QUIC/ GQUIC Traffic," https://www.cellstream.com/reference-reading/tipsandtricks/382-using-wireshark-to-analyze-quic-traffic#!/ccomment-comment=262, Marzo 2018, Último acceso: 14/10/2019.

[22] D. Sanches Gómez, "Desarrollo de un sistema eficiente de análisis del protocolo IEC 60870-5-104 para la detección de anomalías de redes SCADA," Trabajo de Fin de Grado, Grado en Ingeniería de Tecnologías y Sistemas de Telecomunicación. Universidad Autónoma de Madrid, Julio 2019.

A

Apéndice

A.1. Tabla de campos TAG en CHLO y REJ

Tabla A.1: Campos TAG en CHLO y REJ

TAG	Nombre (en inglés)	Breve Descripción	Tipo de dato	Capturado
PAD	Padding	Relleno del paquete.	Numérico	NO
IRTT	Estimated Initial RTT	Estimación probabilística del RTT. 500 ms por defecto.	Numérico	SI
SNI	Server Name Indication	Indica el Hostname.	String	SI
AEAD	Authenticated Encryption Algorithms	Nombres de los algorítmos de cifrados intercambiados.	String	SI
UAID	Client's User Agent ID	Nombre del cliente que realiza la petición.	String	SI
STK	Source Address Token	Prueba de la propiedad de la dirección origen.	String	NO
SNO	Server Nonce	Número aleatorio enviado por el servidor.	Numérico	NO
VER	Version	Version del protocolo.	String	SI
CCS	Common Certificate Sets	Certificados del cifrado.	Byte Array	NO
NONC	Client Nonce	Número aleatorio enviado por el cliente.	Numérico	NO
SCID	Server Config ID	Configuraciones del servidor.	String	NO
TCID	Connection ID truncation	Indica si soporta IDs truncadas	Numérico	NO
PDMD	Proof Demand	Lista de pruebas aceptadas.	String	NO
SMHL	Support Max Header List	Valor máximo soportado por la cabecera.	Numérico	NO
ICSL	Idle Connection State	Valor del cierre de conexión impícito. 30 segundos por defecto.	Numérico	SI
NONP	Client Proof Nonce	Número aleatorio enviado por el cliente.	Numérico	NO
PUBS	Public value	Valores de algoritmos públicos usados por el cliente.	String	NO
MIDS	Max Incoming Dynamic Streams	Cuántos streams simultáneos soporta.	Numérico	NO
SCLS	Silently Close on Timeout	Permiso para cerrado imlícito tras *timeout*.	Numérico	NO
KEXS	Key Exchange Algorithms	Claves de algoritmos intercambiados.	Byte Array	NO
XLCT	Expected Leaf Certificate	Hash del certificado *leaf*.	Byte Array	NO
CSCT	Signed Cert Timestamp	Marca de tiempo del certificado. Normalmente Missing	Numérico	NO
COPT	Connection Options	Opciones pedidas por el cliente o servidor.	String	NO
CCRT	Cache Certificates	Certificados en cache del servidor.	String	NO
PROF	Proof Signature	Firma del servidor para la clave pública de la configuración.	Byte Array	NO
SCFG	Server Config	Mensaje con la respuesta al cliente de la configuración del servidor.	String	NO
RREJ	Reasons for Server Sending	El porqué envía un REJ el servidor.	Numérico	NO
CFCW	Initial session/connection	Número de Bytes por ventana de control de flujo por conexión.	Numérico	NO
SFCW	Initial Stream Flow Control	Número de Bytes por ventana de control de flujo por sesión.	Numérico	NO

A.2. Resultados mostrados por el disector

Tabla A.2: Campos Impresos

Campo impreso	Descripción	Formato/unidades	Default Value
1src_addr	Dirección IP de origen.	IPv4	0.0.0.0
2dst_addr	Dirección IP de destino.	IPv4	0.0.0.0
3src_port	Puerto UDP de origen.	Numérico	0
4dst_port	Puerto UDP de destino.	Numérico	0
5time_first_pkt	Marca de tiempo del primer paquete de la sesión.	Tiempo en epoch	-1
6time_last_pkt	Marca de tiempo del último paquete de la sesión.	Tiempo en epoch	-1
7cid	CID de la sesión.	Numérico	0
8hostname	Nombre del servidor.	String	NULL
9ctos_ip_len_total	Número de Bytes de IP enviados del cliente al servidor.	[Bytes]	0
10ctos_ip_header_len_total	Número de Bytes de IP sin contar la cabecera enviados del cliente al servidor.	[Bytes]	0
11ctos_udp_len_total	Número de Bytes de UDP enviados del cliente al servidor.	[Bytes]	0
12ctos_udp_header_len_total	Número de Bytes de UDP sin contar la cabecera enviados del cliente al servidor.	[Bytes]	0
13ctos_quic_len_total	Número de Bytes de QUIC enviados del cliente al servidor.	[Bytes]	0
14ctos_quic_header_len_total	Número de Bytes de QUIC sin contar la cabecera enviados del cliente al servidor.	[Bytes]	0
15stoc_ip_len_total	Número de Bytes de IP enviados del servidor al cliente.	[Bytes]	0
16stoc_ip_header_len_total	Número de Bytes de IP sin contar la cabecera enviados del servidor al cliente.	[Bytes]	0
17stoc_udp_len_total	Número de Bytes de UDP enviados del servidor al cliente.	[Bytes]	0
18stoc_udp_header_len_total	Número de Bytes de UDP sin contar la cabecera enviados del servidor al cliente.	[Bytes]	0
19stoc_quic_len_total	Número de Bytes de QUIC enviados del servidor al cliente.	[Bytes]	0
20stoc_quic_header_len_total	Número de Bytes de QUIC sin contar la cabecera enviados del servidor al cliente.	[Bytes]	0
21ctos_first_pkn	Valor del PKN primer paquete enviado del cliente al servidor.	Numérico	0
22ctos_pkn_max	Valor del PKN último paquete enviado del cliente al servidor.	Numérico	0
23ctos_pkn_total	Total de paquetes enviados del cliente al servidor.	Numérico	0
24ctos_void	Total de huecos entre paquetes enviados del cliente al servidor.	Numérico	0
25stoc_first_pkn	Valor del PKN primer paquete enviado del servidor al cliente.	Numérico	0
26stoc_pkn_max	Valor del PKN último paquete enviado del servidor al cliente.	Numérico	0
27stoc_pkn_total	Total de paquetes enviados del servidor al cliente.	Numérico	0
28stoc_void	Total de huecos entre paquetes enviados del servidor al cliente.	Numérico	0
29time_first_chlo	Marca de tiempo del primer paquete CHLO enviador por el cliente al servidor.	Tiempo en epoch	-1
30time_first_rej	Marca de tiempo del primer paquete REJ enviador por el servidor al cliente.	Tiempo en epoch	-1
31time_first_ack	Marca de tiempo del primer paquete ACK enviado.	Tiempo en epoch	-1
32time_first_nack	Marca de tiempo del primer paquete NACK enviado.	Tiempo en epoch	-1
33count_chlo	Total de paquetes CHLO.	Numérico	0
34count_rej	Total de paquetes REJ.	Numérico	0
35count_ack	Total de paquetes ACK.	Numérico	0
36count_nack	Total de paquetes NACK.	Numérico	0
37version	Versión de la sesión. Si no es Q043, el disector puede no haber funcionado bien.	String	NULL
38time_connection_close	Marca de tiempo de cerrado de sesión por envío de CONNECTION CLOSE.	Tiempo en epoch	-1
39irtt	Valor del campo IRTT.	[µs]	-1
40rtt_burst	Valor calculado del RTT según las ráfagas de paquetes.	[s]	-1
41total_throughput	Valor de la tasa de transferencia total.	[Bytes/s]	0
42quic_throughput	Valor de la tasa de transferencia de los datos en QUIC.	[Bytes/s]	0
43ctos_loss	Porcentaje de pérdidas de paquetes enviados del cliente al servidor.	%	0
44stoc_loss	Porcentaje de pérdidas de paquetes enviados del servidor al cliente.	%	1
45rttcalculated	Valor calculado del RTT según el envío y recibo de los primeros paquetes CHLO y REJ.	[s]	0
46uaid	Valor del UAID.	String	NULL
47aead	Valor del AEAD.	String	NULL

A.3. Resultados Navegación Web

Tabla A.3: Resultados búsqueda web I

1src_addr	2dst_addr	3src_port	4dst_port	5time_first_pkt	6time_last_pkt	7cid	8hostname
192.168.241.213	172.217.17.14	37062	443	1569319918.807916	1569319934.340549	16984787017389145 30	contributor.google.com
192.168.241.213	172.217.168.162	46705	443	1569319914.078525	1569319934.346646	49816635934947621 20	securepubads.g.doubleclick.net
192.168.241.213	172.217.16.227	39092	443	1569319910.516262	1569319934.347406	99169228393714175 14	www.gstatic.com
192.168.241.213	216.58.211.46	60707	443	1569319912.956208	1569319934.340426	54385961357533490 00	clients1.google.com
192.168.241.213	172.217.17.22	59799	443	1569319916.662600	1569319934.345335	12683113544349302574	i.ytimg.com
192.168.241.213	172.217.17.2	40247	443	1569319914.956650	1569319934.340390	57221231270181248 44	adservice.google.es
192.168.241.213	216.58.201.176	35399	443	1569319926.837027	1569319934.346687	75629868119982802 62	storage.googleapis.com
192.168.241.213	216.58.201.174	37292	443	1569319919.929206	1569319934.344898	16224307096562527947	fundingchoicesmessages.google.com
192.168.241.213	172.217.16.225	47763	443	1569319916.658301	1569319934.347480	43440587660755544 08	yt3.ggpht.com
192.168.241.213	216.58.201.131	41366	443	1569319913.923642	1569319934.347403	14593073076609722238	www.google.es
192.168.241.213	172.217.168.174	58431	443	1569319910.515663	1569319934.340394	13300956353527427344	apis.google.com
192.168.241.213	216.58.201.163	42146	443	1569319911.639103	1569319934.345470	16785629951856198 89	id.google.com
192.168.241.213	172.217.168.166	60222	443	1569319916.380046	1569319934.346684	18099808155939635317	static.doubleclick.net
192.168.241.213	216.58.211.36	53909	443	1569319906.384223	1569319934.347358	10645128350389217056	www.google.com
192.168.241.213	216.58.211.46	34921	443	1569319912.099893	1569319934.346641	19733030161502810 26	ogs.google.com
192.168.241.213	172.217.17.2	53628	443	1569319914.961569	1569319915.041497	37528364384945928 27	adservice.google.com
192.168.241.213	172.217.17.2	33731	443	1569319913.483173	1569319934.345130	17584924478071509109	googleads.g.doubleclick.net
192.168.241.213	216.58.211.33	36890	443	1569319925.872146	1569319934.346817	59008679627857970 95	tpc.googlesyndication.com
192.168.241.213	172.217.16.227	35728	443	1569319907.359980	1569319934.340592	38089982264185601 03	fonts.gstatic.com
192.168.241.213	216.58.201.174	47797	443	1569319919.901123	1569319934.340922	86095425641123164 03	fundingchoicesmessages.google.com
192.168.241.213	216.58.201.141	47884	443	1569319906.245669	1569319934.340348	14633669519472712348	accounts.google.com
192.168.241.213	172.217.168.174	44792	443	1569319913.995067	1569319934.340429	865378127086922498	clients2.google.com

9ctos_ip_len_total	10ctos_ip_header_len_total	11ctos_udp_len_total	12ctos_udp_header_len_total	13ctos_quic_len_total	14ctos_quic_header_len_total
3685	3525	3525	3461	3461	687
15441	13721	13721	13033	13033	9315
4533	4013	4013	3805	3805	809
4027	3787	3787	3691	3691	859
3866	3546	3546	3418	3418	544
4784	4404	4404	4252	4252	1352
3532	3352	3352	3280	3280	488
6702	6202	6202	6002	6002	3036
3576	3376	3376	3296	3296	474
4139	3899	3899	3803	3803	977
16517	12017	12017	10217	10217	4911
3834	3654	3654	3582	3582	772
19248	13808	13808	11632	11632	5778
21515	17075	17075	15299	15299	10023
4051	3771	3771	3659	3659	799
2917	2837	2837	2805	2805	61
27016	23156	23156	21612	21612	16638
5871	5031	5031	4695	4695	1493
3007	2887	2887	2839	2839	75
8356	7056	7056	6536	6536	3078
3705	3525	3525	3453	3453	657
4265	4085	4085	4013	4013	1211
174587	**148727**	**148727**	**138383**	**138383**	**64037**

15stoc_ip_len_total	16stoc_ip_header_len_total	17stoc_udp_len_total	18stoc_udp_header_len_total	19stoc_quic_len_total	20stoc_quic_header_len_total
5563	5403	5403	5339	5339	3929
155129	152249	152249	151097	151097	149334
55127	54287	54287	53951	53951	52473
5170	4970	4970	4890	4890	3476
28033	27573	27573	27389	27389	25949
7319	6919	6919	6759	6759	5325
9917	9697	9697	9609	9609	8147
9479	8839	8839	8583	8583	7125
11657	11437	11437	11349	11349	9875
5197	4937	4937	4833	4833	3413
561221	552741	552741	549349	549349	546780
5041	4881	4881	4817	4817	3407
700986	690646	690646	686510	686510	683655
430329	422449	422449	419297	419297	416771
18133	17773	17773	17629	17629	16199
4193	4113	4113	4081	4081	2679
271234	265174	265174	262750	262750	260531
74523	73343	73343	72871	72871	71337
4241	4141	4141	4101	4101	2697
119743	117643	117643	116803	116803	115151
5157	4997	4997	4933	4933	3523
5458	5298	5298	5234	5234	3824
2492850	**2449510**	**2449510**	**2432174**	**2432174**	**2395600**

Tabla A.4: Resultados búsqueda web II

21ctos_first_pkn	22ctos_pkn_max	23ctos_pkn_total	24ctos_vold	25stoc_first_pkn	26stoc_pkn_max	27stoc_pkn_total	28stoc_vold
1	8	8	0	1	8	8	0
1	86	86	0	1	144	144	0
1	26	26	0	1	42	42	0
1	12	12	0	1	10	10	0
1	16	16	0	1	23	23	0
1	19	19	0	1	20	20	0
1	9	9	0	1	11	11	0
1	25	25	0	1	32	32	0
1	10	10	0	1	11	11	0
1	12	12	0	1	13	13	0
1	225	225	0	1	436	424	6
1	9	9	0	1	8	8	0
1	16	272	1	1	535	517	4
1	222	222	0	1	394	394	0
1	14	14	0	1	18	18	0
1	4	4	0	1	4	4	0
1	193	193	0	1	303	303	0
1	42	42	0	1	59	59	0
1	6	6	0	1	5	5	0
1	65	65	0	1	105	105	1
1	9	9	0	1	8	8	0
1	9	9	0	1	8	8	0

29time_first_chlo	30time_first_rej	31time_first_ack	32time_first_nack	33count_chlo	34count_rej	35count_ack	36count_nack
1569319918.807916	1569319918.834816	1569319918.834816	-1.000000	2	1	2	0
1569319914.078525	1569319914.107379	1569319914.107379	-1.000000	2	1	1	0
1569319910.516262	1569319910.543905	1569319910.543905	-1.000000	2	1	1	0
1569319912.956208	1569319912.984415	1569319912.984415	-1.000000	2	1	1	0
1569319916.662600	1569319916.696096	1569319916.696096	-1.000000	2	1	1	0
1569319914.956650	1569319914.983446	1569319914.983446	-1.000000	2	1	1	0
1569319926.837027	1569319926.869129	1569319926.869129	-1.000000	2	1	2	0
1569319919.929206	1569319919.958201	1569319919.958201	-1.000000	2	1	2	0
1569319916.658301	1569319916.686767	1569319916.686767	-1.000000	2	1	1	0
1569319913.923642	1569319913.949900	1569319913.949900	-1.000000	2	1	1	0
1569319910.515663	1569319910.543867	1569319910.543867	-1.000000	2	1	1	0
1569319911.639103	1569319911.666241	1569319911.666241	-1.000000	2	1	1	0
1569319916.380046	1569319916.408451	1569319916.408451	-1.000000	2	1	1	0
1569319906.384223	1569319906.417163	1569319906.417163	-1.000000	2	1	1	0
1569319912.099893	1569319912.126687	1569319912.126687	-1.000000	2	1	1	0
1569319914.961569	1569319914.990943	1569319914.990943	-1.000000	2	1	1	0
1569319913.483173	1569319913.510874	1569319913.510874	1569319930.759399	2	1	1	1
1569319925.872146	1569319925.903122	1569319925.903122	-1.000000	2	1	1	0
1569319907.359980	1569319907.388440	1569319907.388440	-1.000000	2	1	1	0
1569319919.901123	1569319919.931448	1569319919.931448	-1.000000	2	1	2	0
1569319906.245669	1569319906.273356	1569319906.273356	-1.000000	2	1	1	0
1569319913.995067	1569319914.023549	1569319914.023549	-1.000000	2	1	1	0

37version	38time_cc	39rtt	40rtt_burst	41total_throughput	42quic_throughput	43ctos_loss	44stoc_loss	45rttcalculated	46uaid	47aead
Q043	-1.000000	-1	0.026920	595.391648	297.180779	0.000000	0.000000	0.026900	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	30891	0.028874	8415.678985	7827.513955	0.000000	0.000000	0.028854	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	31127	0.027649	2503.446765	2235.813787	0.000000	0.000000	0.027643	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	29479	0.028284	430.083532	202.719595	0.000000	0.000000	0.028207	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	34080	0.033591	1803.963022	1498.241084	0.000000	0.000000	0.033496	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	-1	0.026880	624.389309	344.463969	0.000000	0.000000	0.026796	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	-1	0.032130	1790.893328	1149.852323	0.000000	0.000000	0.032102	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	-1	0.029066	1122.457382	704.856897	0.000000	0.000000	0.028995	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	48806	0.028523	861.147940	585.046939	0.000000	0.000000	0.028466	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	20542	0.026335	457.114629	214.945718	0.000000	0.000000	0.026258	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	30553	0.028232	24249.507677	23156.231960	0.000000	1.415094	0.028204	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	32782	0.027217	390.859533	184.045294	0.000000	0.000000	0.027138	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	-1	0.028482	40087.299386	38372.955286	0.367647	0.773694	0.028405	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	25123	0.033020	16158.560188	15262.737885	0.000000	0.000000	0.032940	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	26434	0.026824	997.179454	764.066731	0.000000	0.000000	0.026794	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	538213	0.029455	88955.147147	34280.886524	0.000000	0.000000	0.029374	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	27019	0.027724	14296.357669	13285.858035	0.000000	0.000000	0.027701	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	42721	0.030996	9486.385821	8593.843811	0.000000	0.000000	0.030976	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	28600	0.028537	268.637348	102.740443	0.000000	0.000000	0.028460	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	-1	0.030406	8871.245325	8187.717809	0.000000	0.952381	0.030325	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	37990	0.027710	315.433394	148.782621	0.000000	0.000000	0.027687	Chrome/74.0.3729.157 Linux x86_64	AESG
Q043	-1.000000	23915	0.028502	477.897612	247.476548	0.000000	0.000000	0.028482	Chrome/74.0.3729.157 Linux x86_64	AESG

A.4. Resultados de la validación con Tshark en Excel

Tabla A.5: Validación Excel

Campo impreso	Valor obtenido	Validación
1src_addr	192.168.241.144	OK
2dst_addr	216.58.211.34	OK
3src_port	59033	OK
4dst_port	443	OK
5time_first_pkt	1559229684.410359	OK
6time_last_pkt	1559229700.286426	OK
7cid	9167831283005642366	OK
8hostname	securepubads.g.doubleclick.net	OK
9ctos_ip_len_total	3738.000000	OK
10ctos_ip_header_len_total	3558.000000	OK
11ctos_udp_len_total	3558.000000	OK
12ctos_udp_header_len_total	3486.000000	OK
13ctos_quic_len_total	3486.000000	NP
14ctos_quic_header_len_total	712.000000	OK
15stoc_ip_len_total	4671.000000	OK
16stoc_ip_header_len_total	4511.000000	OK
17stoc_udp_len_total	4511.000000	OK
18stoc_udp_header_len_total	4447.000000	OK
19stoc_quic_len_total	4447.000000	OK
20stoc_quic_header_len_total	3037.000000	NP
21ctos_first_pkn	1	OK
22ctos_pkn_max	9	OK
23ctos_pkn_total	9	OK
24ctos_void	0	OK
25stoc_first_pkn	1	OK
26stoc_pkn_max	8	OK
27stoc_pkn_total	8	OK
28stoc_void	0	OK
29time_first_chlo	1559229684.410359	OK
30time_first_rej	1559229684.436193	OK
31time_first_ack	1559229684.436193	OK
32time_first_nack	-1000000	OK
33count_chlo	2	OK
34count_rej	1	OK
35count_ack	1	OK
36count_nack	0	OK
37version	43	OK
38time_connection_close	-1.000000	OK
39irtt	31492	Ok
40rtt_burst	0.026550	Ok
41total_throughput	529.665182	Ok
42quic_throughput	236.141606	NP
43ctos_loss	0	Ok
44stoc_loss	0	Ok
45rttcalculated	0.025834	Ok
46uaid	Chrome/74.0.3729.157 Linux x86_64	Ok
47aead(hex)	47534541	Err