Impacto Funcional del Protocolo QUIC en la Gestión de Conexiones Eficientes en Redes Inalámbricas

Gabriel Alejandro Delgado Villanueva [1][C.I: 29 553 027] and
Arturo Alexander Hung Merentes [2][C.I: 29 621 867]

[1] Universidad Católica Andrés Bello, Escuela de Ingeniería Informática, Caracas, Venezuela, 1020

gadelgado.19@est.ucab.edu.ve / gavd1331@gmail.com

[2] Universidad Católica Andrés Bello, Escuela de Ingeniería Informática, Caracas, Venezuela, 1020

aahung.20@est.ucab.edu.ve / ahmerentes@gmail.com

Abstract. TCP (Transmission Control Protocol) has been a widely adopted transport protocol on the Internet for many years. However, with the continuous evolution of wireless networks and the ever-growing demand for faster and more efficient connections have highlighted certain limitations of TCP, with one of its major drawbacks being the suboptimal behavior of the protocol over networks, particularly in terms of packet recovery time. As internet traffic rapidly expands, these limitations have become more pronounced, making it necessary to explore alternative solutions. In this context, the QUIC (Quick UDP Internet Connections) protocol has emerged as a promising and innovative approach by Google, in order to address these challenges and improve the management of connections. This research focuses on analyzing the QUIC protocol, employing a documentary research methodology. The results cover a detailed explanation of the protocol's design and the implementation of several of QUIC's mechanisms concerning congestion control, reduced round-trip time and security; focusing at the same time on the integration of the protocol in wireless networks to effectively reduce latency and prevent unnecessary retransmissions in the event of data packets.

Keywords: Transport Protocol, Wireless Networks, QUIC, Management of Connections, Congestion Control, Latency

1 Introducción

En la actualidad, Internet se ha convertido en una herramienta fundamental utilizada por el hombre para acceder a distintos servicios y aplicaciones de vanguardia, muchos de ellos desarrollados por líderes en tecnología como Amazon o Google. No obstante, el constante aumento en la cantidad de datos generados y la creciente demanda de transferencias rápidas, seguras y eficientes proponen una serie de obstáculos a la pila (*stack*) HTTPS, donde el protocolo HTTP se transporta sobre TCP y se asegura mediante TLS [1]. A medida que las redes evolucionan, resulta evidente que TCP muestra cada vez más limitaciones en términos de escalabilidad y latencia.

En ese contexto, surge Quick UDP Internet Connections (QUIC), desarrollado por Google en el año 2013 como un protocolo experimental [2], cuyo funcionamiento se sustenta directamente sobre UDP y que en líneas generales soporta la descarga de páginas web mediante la realización de más peticiones en una única sesión TCP, reduciendo con ello el tiempo para el establecimiento de conexiones y una mejor latencia en las comunicaciones. Considerando la posterior estandarización de dicho protocolo en el RFC 9000 [3] por parte de IETF, QUIC constituye un protocolo de transporte que proporciona a las aplicaciones flujos de comunicación mediante el establecimiento de conexiones de baja latencia y migraciones de ruta de red, abarcando con ello medidas de seguridad que garantizan una mejora en las comunicaciones. Básicamente, este protocolo fue diseñado con el objetivo de mejorar la eficiencia de las comunicaciones HTTP en Internet al reducir el tiempo de respuesta, disminuyendo con ello el número de paquetes necesarios para establecer el intercambio de información entre dos equipos. Para esto, utiliza credenciales almacenadas de una conexión previa entre los mismos equipos y elimina los paquetes redundantes o innecesarios del *handshake*, lo que permite que la conexión se defina de manera más inmediata.

En investigaciones previas, se ha reconocido el impacto de las pérdidas en el canal de comunicación, especialmente en redes inalámbricas utilizadas en campos como el Internet de las cosas (IoT), donde tales pérdidas son interpretadas como señales de congestión en la red, lo cual puede resultar en una dimensión subóptima de la misma. Para abordar este problema, se ha utilizado la técnica de corrección de errores Forward Error Correction (FEC) [4], mediante la codificación XOR, la cual mejora la confiabilidad de la capa de enlace al permitir la reconstrucción de paquetes perdidos. Sin embargo, el uso de FEC implica retransmisiones adicionales, lo que reduce los tiempos de espera y mejora el rendimiento del ancho de banda.

Es conveniente destacar que actualmente, QUIC utiliza por defecto el mecanismo de control de congestión TCP CUBIC [4] [5]. Este mecanismo ofrece la ventaja de ajustar la ventana de congestión de manera más escalable y estable en comparación con los mecanismos previos de TCP. No obstante, dado que QUIC tiene un buen desempeño en redes inalámbricas, se ha considerado la posibilidad de utilizar otro mecanismo de control de congestión llamado Bottleneck Bandwidth and Round-Trip Time (BBR) [9], este último el cual propone no considerar la pérdida de paquetes como una señal de congestión, manteniendo así el ancho de banda estable y reduciendo la latencia. Como bien se mencionó con anterioridad, es un protocolo recientemente estandarizado, por lo que aún se mide y experimenta su influencia a futuro en el tráfico de Internet.

El resto del trabajo se estructura de la siguiente manera: en la segunda sección; se detallará el estado del arte, mencionando brevemente la metodología utilizada, y se describen antecedentes o investigaciones previas sobre el *stack* de protocolos TCP/TLS para analizar la arquitectura del protocolo en la gestión de conexiones efectivas en redes inalámbricas; en la sección tres se presentará la estandarización del protocolo QUIC, abarcando características tales como la recuperación ante pérdidas, el control de congestión o el control de flujo; finalmente, en la cuarta sección se presentan las conclusiones.

2 Estado del Arte

En primera instancia, para esta investigación se utilizó la metodología de revisión documental, que de acuerdo a lo planteado por Hart [5] implica la búsqueda sistemática, selección exhaustiva y análisis crítico de documentos relevantes para una

problemática planteada, proporcionando en este caso, una base teórica sólida que permita recopilar y sintetizar la evidencia del funcionamiento del protocolo QUIC para la gestión de conexiones óptimas en redes inalámbricas.

En la actualidad, muchas aplicaciones utilizan el protocolo HTTP. Aunque HTTP/2, estandarizado en febrero de 2015, representa una mejora significativa respecto a su predecesor HTTP/1.1, aún se ejecuta sobre TCP/TLS y, por lo tanto, puede presentar problemas de rendimiento [7] [8]. Algunos de estos problemas incluyen el número de RTT (Round Trip Time) requeridos para la fase de handshake y el bloqueo de la línea principal.

Para abordar estos problemas, Google ha propuesto el protocolo QUIC (Quick UDP Internet Connection) [9], un protocolo a nivel de usuario que se ejecuta sobre el protocolo UDP (User Datagram Protocol) en lugar de TCP. Fue diseño por la empresa de tecnología de información argumentando que su diseño arquitectónico ofrece un tiempo de respuesta más corto (tiempo de carga de la página), lo que resulta en una mejor experiencia para el usuario final

Igualmente, el protocolo de control de transmisión (TCP) es el más utilizado como base para la mayoría de las aplicaciones debido a su capacidad para transportar de manera confiable y eficiente los datos organizándose en un flujo ordenado, lo que permite la identificación de pérdidas de datos y realizar retransmisiones correspondientes. Asimismo, TCP logra implementar mecanismos referentes al control de congestión y el control de flujo, de tal manera que evita la sobrecarga de la red y garantiza una transmisión efectiva. Estos conceptos son fundamentales en la capa de transporte, para que así los protocolos de capa de Aplicación, como HTTP, no tengan que preocuparse por estos y se garantice que los datos lleguen completos y sin pérdidas. Por consiguiente, TCP se emplea ampliamente como protocolo subyacente para transportar datos de HTTP y otros protocolos de nivel de Aplicación.

En el marco del modelo OSI, la transferencia de HTTP sobre TCP se protege adicionalmente mediante TLS [1], una configuración de seguridad ubicada encima de TCP, como se puede apreciar en la Fig. 1. A pesar de su amplio uso en la transmisión de datos, TCP presenta varias desventajas que han sido objeto de estudio y mejora en la comunidad científica y técnica.

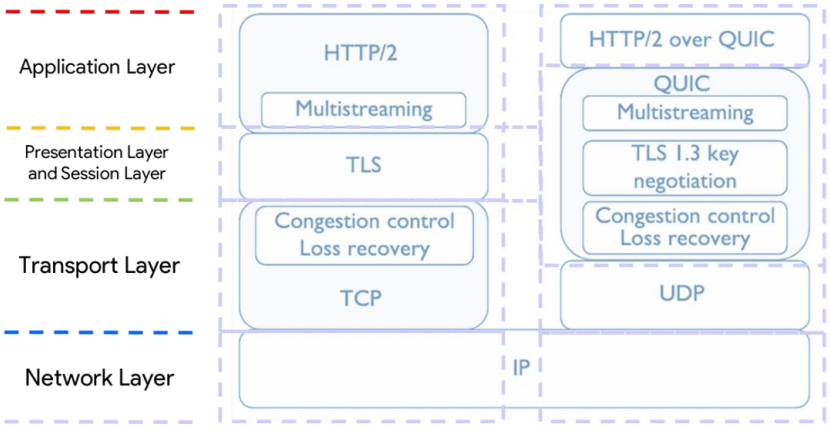

Fig 1: Arquitectura de la red con QUIC, adaptado de [6].

Una de las principales desventajas que presenta TCP es la dificultad de desarrollar y publicar nuevas versiones de implementación de núcleo, ya que este protocolo es integrado a nivel de sistema operativo; es decir, las conexiones de red basadas en TCP se ejecutan en modo kernel, lo cual hace que efectuar cambios en TCP implique a su vez modificar el sistema operativo [10]. Además de ello, presenta inconvenientes en el retardo del handshake, debido a que se necesita por lo menos una ida y vuelta para establecer la conexión exitosa y la inclusión de encriptación por parte de TLS, aumenta aún más dicho retardo. Para solucionar esta problemática, QUIC se ejecuta en el espacio del usuario junto con la implementación del protocolo de transporte UDP, por el hecho de que es un protocolo no orientado a la conexión, siendo así ligero y permitiendo atravesar los middleboxes como firewalls [3]. Muchos firewalls, especialmente en grandes empresas, bloquean protocolos desconocidos, por lo que los paquetes desconocidos podrían ser descartados por middleboxes. En ese sentido, QUIC aplica encriptación a los paquetes de datos para que estos puedan ser transportados por UDP, aumentando la probabilidad de que sean enviados y recibidos en internet de manera fiable.

No obstante, UDP no cuenta con características que se encuentran presentes en TCP en materia de seguridad, por lo que el diseño de QUIC debe incluir en su funcionamiento apartados tales como control de flujo, definición de handshake y la elección de un algoritmo de cifrado y autenticación, con el objeto de que el protocolo sea eficiente y seguro en la entrega de datos. Es importante acotar que QUIC utiliza por defecto el algoritmo de control de congestión TCP CUBIC, el cual se enfoca en ajustar el crecimiento de la ventana de datos en función del tiempo real entre dos eventos de congestión, lo que ayuda a mantener una equidad en los tiempos de ida y vuelta (RTT) [11] entre diferentes flujos de datos que compiten dentro de una misma red. Cuando los RTT son cortos, CUBIC se adapta de manera efectiva para mejorar el rendimiento y la compatibilidad con otros protocolos como TCP [12]. Sin embargo, al momento en el que los RTT sean muy cortos, la tasa de crecimiento de la ventana puede ser más lenta que la de los estándares de TCP, como TCP-SACK; pero aun así termina siendo una característica que mejora la compatibilidad con TCP del protocolo. En general, el uso de TCP CUBIC en QUIC ayuda a mejorar la eficiencia y el rendimiento de los flujos de datos en la red [12], incrementado la tasa de transmisión de manera más agresiva durante períodos de baja congestión y reduciendo esta de manera más gradual durante períodos de alta congestión, maximizando la utilización de la red y adaptándose dinámicamente a los cambios de esta, sin que se creen cuellos de botella o retrasos excesivos.

3 Estandarización

En esta sección se presentan las características más importantes de QUIC, incluyendo aspectos relacionados con la multiplexación, la seguridad, el control de congestión, el control de flujo, el proceso de establecimiento de conexión (handshake) y la recuperación ante pérdida. Si bien, inicialmente QUIC fue desarrollado como una infraestructura monolítica por Google [10], a partir de año 2021 fue estandarizado por la IETF en el RFC 9000 [3], manejando ciertas mejoras con respecto a las características iniciales bosquejadas en su estructuración, referentes a los aspectos anteriormente mencionados.

Como se ha comentado previamente, la Internet Engineering Task Force (IETF) decidió impulsar la estandarización de este protocolo desarrollado por Google, a raíz de las ventajas que ofrecía frente a TCP y HTTP / 2, buscando promover la interoperabilidad y la compatibilidad en múltiples implementaciones, facilitando con

ello la adopción y el despliegue de QUIC en una amplia gama de redes y aplicaciones. Sin embargo, la estandarización del protocolo ha traído consigo la conformación de una versión de QUIC que plantea consigo el añadido de funcionalidades, parámetros y negociaciones. Al representar un protocolo de capa transporte, la comunicación se realiza a través de datagramas UDP que contienen uno o varios paquetes QUIC. Cada paquete QUIC tiene una estructura específica, la cual consta de un encabezado QUIC y una o más tramas (*frames*). Es importante destacar que en una misma versión de QUIC, varios paquetes pueden ser agrupados en un solo datagrama UDP, pero las propiedades invariables sólo se aplican al primer paquete del datagrama. El encabezado y las tramas en el paquete QUIC tienen diferentes tipos y funciones específicas, lo que permite una composición eficiente de la información transmitida, garantizando que los datos sean correctamente interpretados por los puntos finales de QUIC y estableciendo una comunicación fluida y confiable. Existen múltiples tipos de encabezados en este protocolo de transporte, y el paquete QUIC abarca de un encabezado y tramas, como se puede evidenciar en la siguiente figura:

Fig 2: Estructura/formato de un paquete QUIC, adaptado de [10]

De igual forma, es importante mencionar que existen dos (2) cabeceras de paquetes QUIC definidas por la IETF: una cabecera larga, empleada con los paquetes iniciales que permiten el establecimiento de la conexión; y una cabecera corta, la cual es usada luego del establecimiento de la conexión y maneja el resto de paquetes que facilitan el intercambio de información.

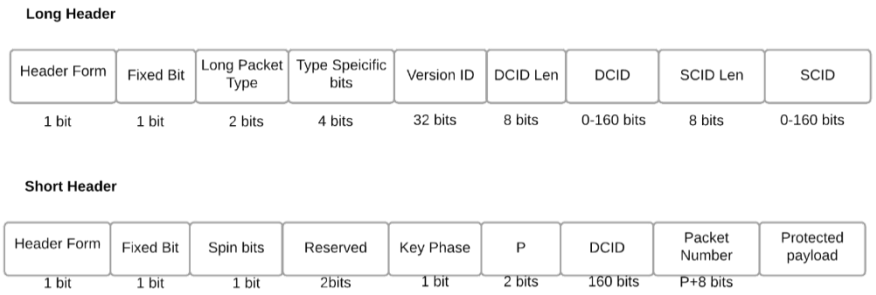

Fig 3: Cabecera QUIC (IETF QUIC - V1), adaptado de [10]

En base a la figura anterior, se detalla la conformación de los campos de cada cabecera.

Cabecera Larga

- **Header Form (HF)**: Identifica el tipo de cabecera de la trama.
- **Fixed Bit (FB)**: Valor que indica si el paquete es válido o no. Si se encuentra definido en cero (0), el paquete es inválido.
- **Long Packet Type (T)**: Indica el tipo de paquete de cabecera larga
- **Type Specific Bits (S)**: Bits específicos para los tipos de paquetes de cabecera larga.
- **Version ID (VID)**: Valor que representa la versión del protocolo con el cual se establece la comunicación
- **Destination Connection ID Length (DCID Len)**: Indica la longitud del campo "Destination Connection ID".
- **Destination Connection ID (DCID)**: Señala el número identificador que el emisor asigna a la conexión.
- **Source Connection ID Length (SCID Len)**: Indica la longitud del campo "Source Connection ID".
- **Source Connection ID (SCID)**: Señala el número identificador que el receptor asigna a la conexión.

Cabecera Corta

- **Header Form (HF)**
- **Fixed Bit (FB)**
- **Spin Bit (SB):** Bit asociado a la latencia.
- **Reserved Bit (RB):** Bit para futuras extensiones del protocolo**.**
- **Key Phase (KP)**: Identifica la llave de protección del paquete.
- **Packet Number Length (P)**: Señala la longitud del campo "Packet Number".
- **Destination Connection ID (DCID)**
- **Packet Number**: Indica el número de paquete.
- **Packet Payload (PP)**: Representa el tamaño máximo de la carga útil; los datos que se transmiten en el paquete después de la cabecera.

En lo que respecta al campo de "Long Packet Type" de la cabecera larga en QUIC, puede responder a un paquete inicial, un paquete 0-RTT, un paquete de handshake o un paquete de reintento, como se ilustra en la Tabla 1.

Tabla 1: Cabecera Larga QUIC. Valores del campo "Long Packet Type"

Tipo	Nombre	Descripción
0x00	Inicial	Transporta los tramas iniciales CRYPTO transmitidas por el cliente y el servidor durante el intercambio de claves y las tramas de reconocimiento de forma bidireccional
0x01	0-RTT	Un paquete 0-RTT envía datos de forma temprana del cliente al servidor antes de que se complete el handshake.
0x02	Handshake	El paquete es para enviar y recibir mensajes de handshake y reconocimientos cifrados entre el servidor y el cliente.
0x03	Retry	El paquete contiene un token de validación de dirección generado por el servidor. Es utilizado por un servidor que desea volver a intentar una conexión.

3.1 Multiplexación

Uno de los principales objetivos con los que fue creado QUIC se ilustra en mejorar el mecanismo de transporte para el protocolo HTTP/2, el cual sufre de un problema muy común conocido como el Bloque de Cabeza de Línea (Head of Line Problem - HoL) [8] [11]. HTTP / 2 resuelve parcialmente el problema con la multiplexación de mensajes en la Capa Aplicación, favoreciendo a la transmisión simultánea de múltiples solicitudes sobre la misma conexión de manera independiente (Figura 4).

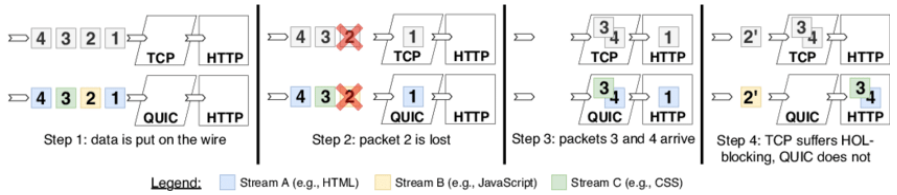

Fig 4: Problema de Bloqueo de Cabeza de Línea en TCP frente QUIC en el contexto de descarga de una página web, adaptado de [6]

A razón de que TCP carece de conocimiento sobre los tres (3) flujos independientes, espera por la retransmisión del paquete 2 (2'). En cambio, QUIC puede enviar inmediatamente los paquetes 3 y 4 a HTTP, donde son procesados antes del paquete mencionado En pocas palabras, esta técnica se vuelve ineficaz cuando ocurre una pérdida de paquetes en el flujo del Protocolo de Control de Transmisión (TCP), ya que todas las corrientes independientes se bloquean hasta la retransmisión del paquete perdido. Es decir, si un solo paquete se cae, o se pierde en la red en algún lugar entre dos (2) puntos finales que se comunican por medio HTTP/2 [13][14], significa que toda la conexión TCP, la cual se encuentra en serie o en secuencia (Figura 5A), se detiene mientras el paquete perdido se retransmite y vuelve a dirigir hacia el destino, teniendo que esperar el resto de paquetes hasta la resolución de dicha eventualidad. Básicamente, el problema surge porque el TCP subyacente no diferencia entre mensajes/corrientes independientes del protocolo de Capa de Aplicación. Considerando que QUIC se encuentra soportado por UDP, debe proporcionar sus propios mecanismos de entrega confiable en la red, aprovechando a su vez las características de máximo esfuerzo (*Best Effort*) de este protocolo orientado a la no conexión para permitir el envío eficiente y veloz de datos. Por lo tanto, QUIC ofrece la entrega confiable y ordenada de múltiples flujos de datos concurrentes al aplicar multiplexación en una única conexión (Figura 5B), evitando el retardo en la entrega de datos en otros flujos, debido a que UDP no está vinculado a la envío secuencial de datos en un solo flujo. De acuerdo al último borrador del protocolo principal, los flujos se identifican mediante un ID de flujo único y el orden de entrega de los datos se maneja con desplazamientos de flujo.

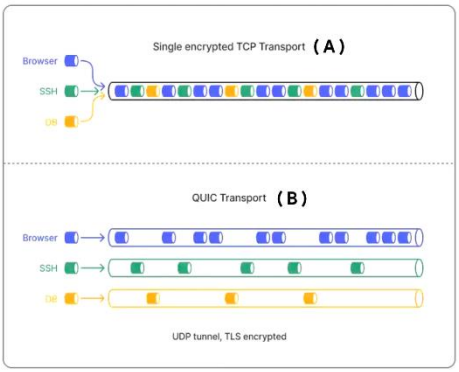

Fig 5: Comparativa del funcionamiento de los protocolos de transporte TCP (A) y QUIC (B) [6]

3.2 Control de Congestión

El protocolo QUIC, en respuesta a la alta congestión de red que se produce en entornos donde se envían una gran cantidad de paquetes en un corto periodo de tiempo, utiliza un algoritmo de control de congestión para regular la tasa de red y disminuir la carga de la misma. La implementación de este algoritmo permite manipular la cantidad de datos transmitidos en un momento dado, evitando así el problema en cuestión y mejorando el rendimiento del sistema. Una de las grandes ventajas que presenta este protocolo, es que posee la capacidad de ser altamente ajustable para que cualquier algoritmo de control de congestión pueda ser probado y experimentado, con el fin de encontrar aquel que sea más óptimo y permita efectuar el trabajo que se desea.

En dicho periodo de prueba, un grupo de desarrolladores de Google utilizaron CUBIC como controlador de congestión [10], en donde se logró evidenciar una gran característica que lo situó como algoritmo de congestión ideal para QUIC. La principal característica que se observó fue en la reproducción de vídeos en dispositivos de escritorio y dispositivos móviles, donde los clientes no-QUIC utilizaban dos conexiones TCP al servidor de video para obtener datos de audio y video. Las conexiones no se designan como tal en conexiones de audio o video, sino que cada fragmento emplea de forma arbitraria una de las dos conexiones disponibles. En cuanto al entorno de QUIC, estos flujos de audio y video son enviados por dos flujos en una única conexión. Para lograr equidad en tráfico similar al uso de TCP, se utilizó una variante de mulTCP [12] de CUBIC en la fase de prevención de congestión. El control de flujo en QUIC es una característica fundamental que garantiza un equilibrio en la transmisión de datos y evita la sobrecarga del receptor, regulando la velocidad de recepción de datos para asegurarse de que el receptor pueda manejarlos adecuadamente.

3.3 Control de Flujo

QUIC implementa dos niveles de control de flujo para lograr un rendimiento óptimo: a nivel de flujo y a nivel de conexión. En lo que respecta a nivel de flujo, se limita la cantidad de datos que cada canal individual puede ocupar en el búfer de recepción, evitando que uno en específico acapare todos los recursos y garantice que

otros también puedan enviar y recibir datos de manera eficiente. En lo que se refiere a nivel de conexión, se controla la cantidad total de datos que la conexión puede recibir en su búfer de recepción, haciendo que no ocurra una sobrecarga general en la conexión y asegurando una distribución equitativa de recursos entre los diferentes flujos. En líneas generales, es un mecanismo crucial para evitar los bloqueos y retrasos innecesarios, para de esta manera establecer un rendimiento efectivo en la transmisión de datos. Al gestionar la velocidad de recepción, se optimiza el uso de los recursos disponibles y se mejora la eficiencia en la comunicación.

3.4 Seguridad

Como se indicó en un inicio, el primer lanzamiento del desarrollo de QUIC fue alrededor del año 2013 por parte de Google. El equipo de investigación de la compañía buscó implementar su propia criptografía para QUIC con el objetivo de proporcionar una carga útil (datos) sin demora en el viaje de ida y vuelta. Dado que TLS 1.3 se plantea como estándar, la IETF decidió utilizarlo como capa de seguridad para QUIC, dado que este proporciona un handshake de 0-RTT a servidores conocidos y, por lo tanto, se ajusta al objetivo de QUIC de reducir la latencia del handshake.

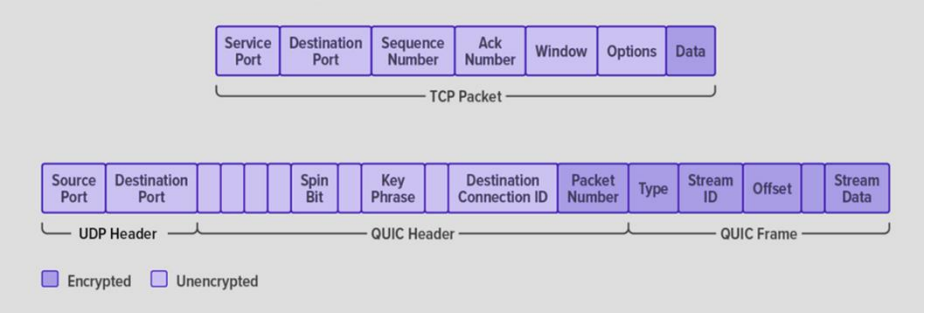

Fig 6: Comparativa de cifrado de metadatos TCP+TLS y QUIC [2]

Tomando en cuenta la Fig. 6, se ilustra cómo QUIC cifra más metadatos potencialmente sensibles que TCP+TLS, siendo estos encriptados utilizando TLS 1.3.

Al abarcar el cifrado de paquetes, evita que pueda haber replicación de ataques de red como *packet sniffing* o *man-in-the-middle* [11] [12], incluso pudiendo detenerse estos. De acuerdo con la implementación actualmente probada en Chromium [6], un servidor QUIC requiere contar con un certificado válido y una clave privada en formatos compatibles para su correcto funcionamiento. Así pues, durante el handshake, debe haber integración de la capa de seguridad TLS incluso antes de que se haya realizado el intercambio de claves, pues los paquetes que definen el inicio de la conexión deben estar encriptados. Es por ello que durante dicho proceso, el cliente debe autenticar la identidad del servidor, implicando que hasta para un servidor de prueba, es necesario instalar un certificado para que pueda operar funcionalmente. Esto permite que QUIC puede establecerse como un protocolo de transporte íntegro, confiable y seguro en la transmisión y recepción de datos, teniendo un efecto positivo en la tendencia de uso global de HTTPS y, por lo tanto, asegurando y autenticando los servidores web registrados en Internet.

3.5 Establecimiento de la conexión (*Hanshaking)*

El protocolo QUIC se ha convertido en una alternativa prometedora para el establecimiento de conexiones seguras y de alto rendimiento en Internet. Una de las características clave de QUIC es su capacidad para cifrar una mayor cantidad de metadatos sensibles en comparación con TCP+TLS. Para comprender mejor el funcionamiento del protocolo en temas de confidencialidad y la autenticación en la comunicación, es fundamental examinar los mensajes utilizados en el protocolo y el contenido que contienen. En este sentido, la Tabla 2 expone una descripción de los mensajes principales utilizados en QUIC y su contenido asociado, que de forma general desempeñan un papel crucial en el establecimiento de la conexión segura entre el cliente y el servidor bajo el protocolo contemplado.

Tabla 2: Mensajes principales en el protocolo QUIC

Mensaje	Significado de la Abreviatura	Descripción
CHLO	Client Hello	Mensaje enviado por el cliente para indicar al servidor su intención de establecer una conexión.
SHLO	Server Hello	Mensaje enviado por el servidor al cliente para indicar su intención de establecer una conexión.
SCFG	Server Config	Mensaje que contiene la información de configuración del servidor, como la clave pública del algoritmo de largo plazo Diffie-Hellman utilizado para encriptar el paquete. Contiene la clave pública del algoritmo Diffie-Hellman y posiblemente otros datos de configuración.
STK	Source Address Token	Bloque encriptado y autenticado que contiene la dirección IP pública del cliente.
REJ	Reject	Mensaje enviado por el servidor al cliente cuando se necesita más información o cuando el mensaje CHLO recibido no es válido. Contiene STK, SCFG, cadena de confianza con certificados del servidor, firma del SCFG con clave privada, y posiblemente otros datos.

De acuerdo a lo pautado por Langley [10] cuando un cliente se comunica con un servidor, puede almacenar información sobre ese servidor y manipularla en conexiones futuras con el mismo origen, permitiéndole utilizar el handshake de 0-RTT en conexiones posteriores con el mismo origen. El handshake de 1-RTT es posible porque las claves de transporte y criptográficas se solapan en la misma capa. De igual forma, el handshake de 0-RTT también es factible gracias a que TLS 1.3 proporciona un handshake de 0-RTT y se utiliza como capa de seguridad en QUIC. A razón de que muchas de las conexiones de red se realizan a los mismos servidores que se contactaron anteriormente, el handshake de 0-RTT permite que un cliente envíe datos sin repetir el intercambio criptográfico o de claves de transporte, distinguiendo así el protocolo entre el handshake de 1-RTT y el handshake de 0-RTT. El handshake criptográfico minimiza la latencia de la comunicación establecida al utilizar las credenciales conocidas del servidor en conexiones repetidas.

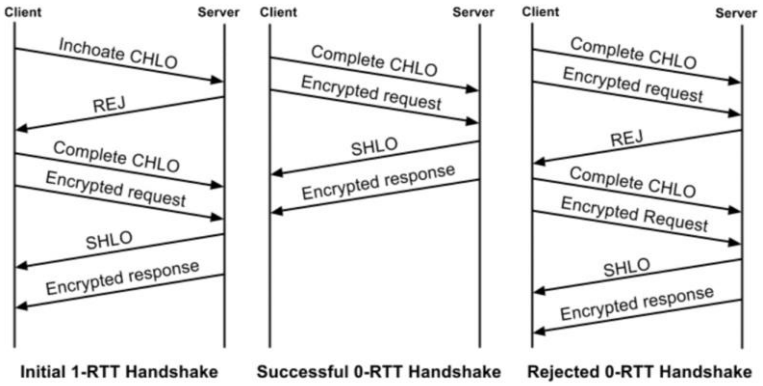

Fig 7: Cronología del handshake inicial de 1-RTT de QUIC, un handshake exitoso de 0-RTT posterior y un handshake de 0-RTT fallido [6]

Considerando la Fig. 7, se describen a continuación los tres (3) tipos de handshake involucrados:

- **Handshake inicial de 1-RTT**: En esta etapa, cuando un cliente establece una conexión con un servidor QUIC por primera vez, se realiza un handshake completo de 1-RTT. Esto implica una serie de intercambios de mensajes entre el cliente y el servidor para establecer las claves criptográficas y autenticarse mutuamente. Una vez completado con éxito, la conexión queda establecida y las comunicaciones pueden comenzar.

- **Handshake exitoso de 0-RTT posterior:** En conexiones posteriores entre el mismo cliente y servidor, si el cliente ha almacenado información previa sobre el servidor y se cumplen ciertas condiciones de seguridad, se puede realizar un handshake de 0-RTT. En este caso, el cliente puede enviar datos al servidor sin tener que repetir todo el proceso de intercambio de claves y autenticación. Esto permite una conexión más rápida y eficiente.

- **Handshake de 0-RTT fallido:** En algunas situaciones, un handshake de 0-RTT puede fallar, si hay cambios en la configuración de seguridad, si la información almacenada del servidor no es válida o si se detectan riesgos de seguridad. En este caso, el cliente debe realizar un handshake completo de 1-RTT nuevamente para establecer una conexión segura.

3.6 Recuperación ante pérdida

TCP utiliza números de secuencia para garantizar la integridad de los datos y el orden de entrega en el receptor. Sin embargo, el problema de "ambigüedad deretransmisión" surge cuando un segmento TCP retransmitido lleva los mismos números de secuencia que el paquete original, lo que dificulta que el receptor de un

ACK de TCP determine si el ACK se envió en respuesta a la transmisión original o la retransmisión, lo que ocasiona que la detección de pérdida en TCP se realice a través de tiempos de espera de costos elevados. Para resolver este problema, cada paquete QUIC lleva un nuevo número de paquete exclusivo [11], incluso los que transportan datos retransmitidos, lo que evita la necesidad de un mecanismo separado para distinguir entre la ACK de una retransmisión y la de una transmisión original. De esta manera, los desplazamientos en los marcos de flujo de QUIC se utilizan para el orden de entrega, separando las dos funciones que TCP conlleva, lo cual provoca que el número de paquetes en QUIC representen un orden temporal explícito y permita una detección de pérdida más simple y precisa que en TCP. Conviene resaltar que las confirmaciones de QUIC incluyen información relacionada al tiempo que tarda un paquete en ser confirmado después de su recepción. Lo anterior, de manera conjunta con los números de paquete que aumentan de manera constante, permite medir de forma precisa el tiempo que tarda un paquete en ir y volver a través de la red, consiguiendo detectar la presencia de paquetes perdidos y ajustar en consecuencia, la transmisión de datos.

4 Conclusiones

La reducción de la latencia es un factor crucial en la gestión eficiente de conexiones en redes inalámbricas, y el protocolo QUIC ha demostrado tener un impacto significativo en este sentido. Al implementarse, se logra una disminución notable en el tiempo de respuesta de las conexiones, lo que mejora la experiencia del usuario y permite una transferencia más rápida de datos. Igualmente, los beneficios de este protocolo se ven materializados con el uso de UDP como capa de transporte subyacente, otorgando mayor flexibilidad y eficiencia en la entrega de datos, contribuyendo así a una mayor velocidad y confiabilidad de las conexiones.

En lo que concierne a la arquitectura funcional de este protocolo, en él se integran mecanismos de multiplexación, seguridad, control de congestión, control de flujo, establecimiento de conexión y recuperación de pérdidas en un nivel superior al de UDP, lo que optimiza el rendimiento de las conexiones y minimiza los efectos negativos referentes a la pérdida de paquetes. Todo ello, consigue que QUIC se ejecute en espacio de usuario, haciendo que aportes relacionados con mejoras en la latencia y la eliminación del Bloqueo de Cabeza de Línea lo conviertan en un estándar propuesto para conexiones eficientes en el Internet actual.

References

1. E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, RFC 8446, DOI 10.17487/RFC8446, agosto 2018, https://www.rfc-editor.org/info/rfc8446.

2. M. Yosofie and B. Jaeger, "Recent Progress on the QUIC Protocol," Chair of Network Architectures and Services, Department of Informatics, Technical University of Munich, Germany, Seminar IITM WS 18/19, Network Architectures and Services, (2019).

3. J. Iyengar y M. Thomson, “A UDP-Based Multiplexed and Secure Transport”, IETF, Internet draft, draft-ietf-quic-transport-19, (2019).

4. F. Fernández, "Mecanismos de control de congestión y errores en el protocolo QUIC", Escuela Técnica Superior de Ingenieros Industriales y De Telecomunicación - Universidad de Cantabria, (2019).

5. C. Hart, "Doing a Literature Review: Releasing the Social Science Research Imagination," Sage Publications, (1998).

6. A. Langley et al., “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” in Proceedings of the Conference of the ACM Special Interest Group on

7. Data Communication, ser. SIGCOMM ’17. New York, NY, USA: ACM, pp. 183–196. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098842, (2017)

8. S. Lázaro, “Evaluación del Uso del Protocolo QUIC en Internet”, Escuela de Ingeniería en Tecnologías de Telecomunicaciones - Universidad Carlos III de Madrid, (2019).

9. A. Cidon, M. B. Taylor, S. Katti, L. E. Li, and I. Stoica, "SCUBIC: A New TCP-Friendly High-Speed TCP Variant," in Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, (2008) Available: http://www.cs.ucr.edu/~jiasi/teaching/cs204_spring16/papers/cubic08.pdf, (2007)

10. Jain, R. "QUIC - A New Internet Transport Protocol", Washington University in St. Louis, Department of Computer Science and Engineering, [Online]. Available; https://www.cse.wustl.edu/~jain/cse570-21/ftp/quic/index.html (2021)

11. Klingler, F., Dressler, F., & Sommer, C. The impact of head of line blocking in highly dynamic WLANs. IEEE Transactions on Vehicular Technology, 67(8), 7664-7676, (2018)

12. S. Chen, S. Jero, M. Jagielski, A. Boldyreva, and C. Nita-Rotaru, "Secure communication channel establishment: TLS 1.3 (over TCP fast open) vs. QUIC," in Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS), Luxembourg, Sept. 23-27, 2019, pp. 404-426, (2019).

13. Cook, S., Mathieu, B., Truong, P., & Hamchaoui, I. QUIC: Better for what and for whom ?. In 2017 IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE (2017).

14. Kumar, P., & Dezfouli, B. “Implementation and analysis of QUIC for MQTT”. Computer Networks, 150, 28-45. (2019).