JITEL 2021

LIBRO DE ACTAS

XV Jornadas de Ingeniería Telemática

A CORUÑA 2021

ISBN: 978-84-09-35131-2

Editores:
Victor Manuel Carneiro Díaz
Laura Victoria Vigoya Morales

XV Jornadas de Ingeniería Telemática – JITEL 2021


XV Jornadas de Ingeniería Telemática.
JITEL 2021.
Universidad de A Coruña.

Actas de las XV Jornadas de Ingeniería Telemática (JITEL 2021), A Coruña (España), 27-29 de octubre de 2021.

Evolución del Stack IoT: MQTT sobre QUIC

Fátima Fernández[1], Mihail Zverev[1], Pablo Garrido[1], José R. Juárez[1], Josu Bilbao[1], Ramón Agüero[2]
[1]Ikerlan Technology Research Centre, Basque Research Technology Alliance (BRTA), Arrasate/Mondragon, España;
{ffernandez, mzverev, pgarrido, jrjuarez, jbilbao}@ikerlan.es
[2]Dpto. de Ingeniería de Comunicaciones, Universidad de Cantabria, Santander, España;
ramon@tlmat.unican.es

En este trabajo se analiza el rendimiento de QUIC como solución de transporte para dar soporte a servicios IoT industriales basados en Message Queuing Telemetry Transport (MQTT). QUIC fue desarrollado por Google para solucionar las limitaciones que presenta el protocolo de transporte dominante, Transmission Control Protocol (TCP). Para estudiar su comportamiento, se han emulado escenarios con características propias a estos entornos industriales, incluyendo QUIC como protocolo de transporte para MQTT y comparándose esta combinación frente a la solución tradicional basada en la pila TCP/TLS/MQTT. Para emular distintas tecnologías y condiciones de red se han empleado contenedores LXC de Linux a modo de dispositivos IoT mediante el simulador de eventos ns-3. Los resultados obtenidos ponen de manifiesto que QUIC podría ser una alternativa interesante como protocolo de transporte en escenarios IoT.

***Palabras Clave*—QUIC; Message Queuing Telemetry Transport (MQTT); Industria 4.0; IoT Industrial (IIoT); Redes Inalámbricas; Entorno de Simulación.**

I. INTRODUCCIÓN

Actualmente, la industria y las empresas están inmersas en la cuarta revolución industrial, denominada Industria 4.0. Este término fue introducido en 2011 por el gobierno alemán, y surgió como una alternativa novedosa hacia la digitalización de la industria en base a cuatro principios: interconexión, transparencia de la información, decisiones descentralizadas y mantenimiento predictivo [1]. Esta revolución trae consigo un aumento de la productividad gracias, entre otras cosas, a la recogida y análisis de datos en tiempo real [2].

Una de las tecnologías clave detrás de esta transformación digital de la industria es el paradigma basado en el Internet de las cosas (*Internet of Things*, IoT), el cual habilita la conexión de dispositivos industriales para recolectar y analizar datos en tiempo real, monitorizar sistemas, intercambiar información, y analizar el entorno industrial [2]. Sin embargo, esta revolución industrial ha desembocado en la necesidad de disponer de nuevos servicios y aplicaciones con estrictos requisitos, como comunicaciones de baja latencia, disponibilidad y fiabilidad en las redes así como reducción en los costes además de ser eficientes energeticamente, canales seguros y la conservación de la privacidad de los datos [3].

Para monitorizar el entorno industrial, se necesita un gran despliegue de sensores y dispositivos conectados entre sí, lo que podría conllevar costes elevados. Además su ubicación en localizaciones diversas y áreas aisladas implica la necesidad de baterías con una vida útil elevada. Las comunicaciones con baja latencia son esenciales para muchas aplicaciones industriales, ya que pueden necesitar de respuestas rápidas para habilitar un proceso de fabricación determinado o garantizar la seguridad de los elementos involucrados. Además, una baja latencia resulta indispensable para habilitar funciones de control remoto en escenarios industriales reales.

MQTT está ganando popularidad en el mundo IoT, y se está convirtiendo en el protocolo de aplicación de facto [4], [5]. Esto se debe principalmente a su fácil integración en los dispositivos y al buen comportamiento que ofrece. Tradicionalmente, funciona sobre el protocolo de transporte TCP [6], que como es sabido ofrece un servicio orientado a la conexión. Sin embargo, TCP no es capaz de adaptarse a la velocidad a la que evoluciona la tecnología, dejando en evidencia la cantidad de desventajas que sufre [7], especialmente las relacionadas con la osificación de los protocolos de internet. Por otro lado, en redes donde la probabilidad de pérdida es alta, el comportamiento de TCP se ve perjudicado aumentando así el retardo de las comunicaciones [8]. Por lo tanto, TCP no es capaz de garantizar el bajo retardo que requieren algunas aplicaciones IIoT [9]. Para mejorar el comportamiento ofrecido por la capa de transporte, se han desarrollado varias alternativas, muchas de ellas pequeñas modificaciones o actualizaciones de TCP, como pueden ser STCP [10], Real-Time TCP [11] o Network Coded TCP [12], entre otras. QUIC aparece como una alternativa más disruptiva, siendo un protocolo de transporte que tiene como objetivos principales reducir la latencia en el establecimiento de la conexión, desarrollo de nuevas características y dotar de seguridad a las comunicaciones habituales en aplicaciones HTTP [13].

En este trabajo se propone la utilización de QUIC como protocolo de transporte para tráfico MQTT, analizando los beneficios temporales de esta combinación en entornos IoT reales, comparándola con las soluciones más empleadas actualmente. Por lo tanto, las principales contribuciones de este trabajo son las siguientes:

- Integración y optimización del socket de QUIC en las implementaciones del broker y cliente de MQTT en el lenguaje de programación GO.
- Código *open-source* en un repositorio público de *github*.
- Análisis del comportamiento de MQTT sobre QUIC, utilizando contenedores de Linux y el simulador de eventos ns-3 para emular distintas tecnologías inalámbricas.

La estructura del artículo es la siguiente: la Sección II describe los antecedentes y el trabajo relacionado. La Sección III ilustra el proceso de implementación de MQTT y QUIC en GO, mientras que la Sección IV explica la configuración de las redes y las conexiones que se utilizan para llevar acabo los experimentos. La Sección V muestra los escenarios y experimentos, comenta los resultados obtenidos y compara el esquema propuesto con la solución tradicional TCP/TLS/MQTT. Por último, la Sección VI recoge las conclusiones finales, así como líneas futuras de investigación.

II. ESTADO DEL ARTE

QUIC es un protocolo de transporte desarrollado originalmente por Google Inc. [13] y recientemente estandarizado por el IETF [14][1]. Además de abordar algunas de las limitaciones de TCP, QUIC proporciona algunos beneficios adicionales que pueden ser relevantes para los escenarios de IIoT, ya que es capaz de asegurar latencias más bajas y, a su vez, ofrecer un servicio fiable y seguro. Por su parte, MQTT es uno de los protocolos de aplicación más populares en el ámbito de las redes IIoT. Esta sección describe el funcionamiento básico de los protocolos MQTT y QUIC, para así entender la motivación de combinar ambos y el impacto que dicha combinación puede llegar a tener en el entorno IoT industrial.

A. MQTT

MQTT [4] es un protocolo basado en el modelo de publicación-suscripción. Gracias a su programación y al bajo ancho de banda de red requerido, ha sido ampliamente utilizado para conectar pequeños dispositivos en una variedad de industrias desde 1999. La versión 3.1.1 fue presentada en 2014 por IBM, y fue estandarizada por ISO y OASIS. Además, la versión 5.0 se ha estandarizado, aunque la versión 3.1.1 sigue siendo la más ampliamente usada [5].

En MQTT existen tres roles: *subscriber*, *publisher* y *broker*. Los *publisher* suelen ser pequeños sensores que generan (perciben/miden) información y la publican en un *broker* común con *topics* específicos. El *subscriber* consume los datos producidos por los *publisher*, de manera que recibe todos los mensajes que haya enviado cualquier *publisher* sobre el *topic* concreto al que está suscrito. Tanto los *publisher* como los *subscribers* pueden considerarse clientes en la topología de red correspondiente. El elemento clave de MQTT es el servidor *broker*, que gestiona las suscripciones. Todos los mensajes publicados en la red se envían al *broker*, que se encarga de distribuir la información a los *subscribers* correspondientes. El *broker* también tiene en cuenta los distintos niveles de calidad de servicio (QoS de sus siglas en inglés) para los clientes, así como las posibles retransmisiones. Aunque los clientes sólo interactúan con un *broker*, el sistema puede contener varios servidores que intercambian datos sobre los *topics* de sus *subscribers* actuales.

Una de las principales ventajas de MQTT es el aislamiento que consigue entre *publishers* y *subscribers*. Esto facilita la implementación de los distintos roles de cliente en dispositivos de baja capacidad computacional, pudiendo interactuar entre sí mediante MQTT. Otro elemento positivo es que el *publisher* puede enviar nuevo contenido siempre que esté disponible, desvinculando así la relación temporal entre el interés de un nodo y la publicación de la información.

B. QUIC

Los dos retos que pretende abordar QUIC, y que afectan principalmente al tráfico web, son la minimización de la latencia para una mejor experiencia de usuario, y proporcionar comunicaciones seguras [15], [16]. En la Figura 1 se muestra la pila de protocolos propuesta con QUIC, comparándola con la solución tradicional TCP/TLS transportando tráfico HTTP.

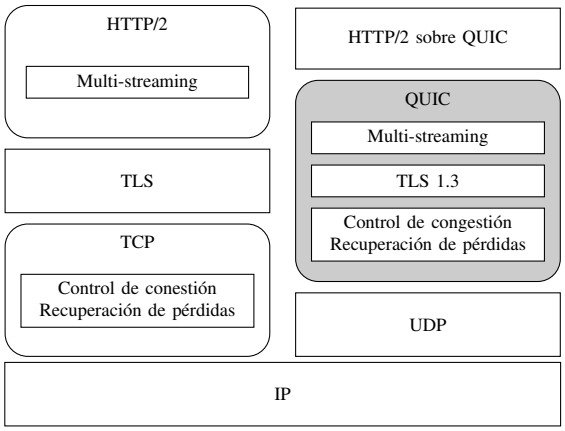


Fig. 1. Arquitectura tradicional de la capa de transporte con TCP/TLS frente a la propuesta con QUIC.

QUIC reduce la latencia en el *handshake*, estableciendo una conexión segura con el protocolo Transport Layer Security (TLS), versión 1.3, en solo un Round Trip Time (RTT), o incluso con cero RTT, si los extremos de la comunicación se han conectado previamente [16]. Por su parte, TCP implementa un intercambio de mensajes inicial

[1]Los RFC de QUIC han sido publicados después de finalizar los experimentos descritos en este trabajo. Por tanto, la versión de QUIC a la que se refiere el artículo es el draft 27, que es la que se empleó a lo largo de todo el trabajo.

denominado *3-way handshake* lo que hace que siempre aparezca un retardo inicial (en el primer segmento de datos) de un RTT [6]. Además, TCP necesita el protocolo TLS para encriptar los datos y poder garantizar así comunicaciones seguras. El establecimiento de la conexión de TLS 1.2 acarrea 2 RTT [17] y la versión 1.3 1 o 0 RTT [18]. Así, al utilizar TLS sobre TCP daría lugar a un *handshake* global de 3 o 2 RTT, respectivamente.

En una conexión QUIC la información está organizada en *streams*, evitando así el retardo causado por la pérdida de paquetes iniciales que bloqueen el resto. Cuando un paquete se pierde, solo los *streams* con datos en dicho paquete se bloquean esperando a la retransmisión del mismo mientras el resto continua [19]. También cabe destacar que QUIC reduce la latencia mediante sus mecanismos de detección de pérdida, incluyendo *early retransmits* y *tail loss probes* [19].

La principal diferencia con los mecanismos de detección de pérdidas utilizados por TCP es que QUIC no retransmite todos los paquetes con el mismo número de secuencia. QUIC, además de utilizar reconocimientos para confirmar la correcta recepción de los paquetes, usa un margen temporal para detectar pérdidas de cola (*probe timeout*). Para aplicar los mecanismos de recuperación que implementa QUIC se distinguen los siguientes supuestos de pérdida de paquete [19]:

- Si no ha llegado su ACK y fue enviado antes de otro paquete posterior ya reconocido.
- Si se mandó previamente con un margen temporal de $9/8 \cdot$RTT o su número de secuencia es 3 veces más pequeño que el del último paquete confirmado.
- Si se alcanza el *probe timeout* desde su envío y no hay ningún paquete posterior reconocido.

El tráfico TCP en Internet se supervisa y, en su caso, se mejora a través de *middleboxes*. Estos dispositivos analizan el contenido de los paquetes a nivel de transporte, y los modifican para conseguir un comportamiento óptimo de la red, por ejemplo adecuando la retransmisión de paquetes [20]. Es importante resaltar que cualquier mejora que se quiera hacer en TCP conlleva una actualización de los *middleboxes* correspondientes ya que, en caso contrario, estos descartarían los segmentos TCP que no puedan procesar. Esto dificulta la actualización de TCP, ya que se implementa en el *kernel* de estos equipos.

QUIC fue diseñado para suplir la rigidez que presenta TCP a la hora de introducir mejoras [13], [20]. Al estar implementado sobre UDP, los *middleboxes* consideran los paquetes QUIC como parte del *payload* de los datagramas UDP. La encriptación de la cabecera y *payload* de QUIC previene la interferencia de los *middleboxes* en el tráfico QUIC en el futuro. La decisión de diseñar e implementar QUIC a nivel de usuario le confiere una mayor flexibilidad, con mayor libertad en términos de capacidad computacional, más interacción con servidores y facilita considerablemente la actualización del protocolo [13]. Sin embargo, este último punto no es de obligado cumplimiento, por lo que QUIC también podría integrarse en el *kernel* para mejorar todavía más su rendimiento [21].

QUIC incluye un intercambio de mensajes donde se negocia la versión a utilizar, habilitando la coexistencia de diferentes modificaciones del mismo. Esto simplifica la actualización del protocolo, y habilita su ampliación y optimización. Por ejemplo, QUIC podría ser ampliado para incluir las demás extensiones como *plugins* permitiendo así que un *endpoint* de la red comparta la funcionalidad que otro no tenga [22].

C. QUIC en entornos IoT

QUIC se está empezando a incluir en las pilas de protocolos para soluciones IoT [23], [24] aunque todavía no existen muchos trabajos que aborden su evaluación en estos escenarios. Liri et al. valoraron QUIC como protocolo IoT en [25], revelando que QUIC, como sustituto de protocolos IoT más tradicionales, no proporciona el rendimiento del protocolo Constrained Application Protocol (CoAP) [26]. Sin embargo, el comportamiento de QUIC en entornos con cierta inestabilidad es comparable al de MQTT-Sensor Networks, variante de MQTT para dispositivos con baja capacidad. Los autores sugieren que una implementación de QUIC más simplificada y optimizada podría ser una alternativa frente a CoAP.

Kumar y Dezfouli estudiaron el comportamiento de la implementación de QUIC creada por Google en escenarios IoT [27]. Compararon el comportamiento de MQTT sobre QUIC y TCP en entornos de simulación acotados, usando dispositivos de baja capacidad computacional, Raspberry Pi 3B. El análisis se basó en el estudio de la sobrecarga de paquetes en el establecimiento de la conexión, el efecto en la latencia al introducir pérdidas aleatorias de paquetes, el uso de la memoria y procesador cuando uno de los *endpoints* rompe la conexión, y el rendimiento durante la migración de la conexión. Sus resultados muestran que QUIC supera a TCP en varios aspectos, e identifican algunas características que deberían mejorarse para un mejor rendimiento en escenarios IoT.

Lars Eggert ha analizado recientemente la viabilidad de integrar QUIC en equipos IoT [28]. En un primer estudio, usó dispositivos más optimizados que los que se utilizaron en [27]: Particle Argon y ESP32-DevKitC V4. Para disminuir el uso de memoria de estos dispositivos de baja capacidad, eliminó algunas de las funcionalidades de QUIC que podían considerarse poco prácticas para entornos IoT. Después de evaluar el uso de memoria y el consumo de energía de estos dispositivos, utilizando esta implementación optimizada de QUIC, se concluye [28] que es una buena opción para dispositivos *edge*.

En este trabajo se analiza el comportamiento de QUIC tal y como se define en [15] sobre escenarios IoT. En lugar de estudiar la capacidad de adaptación de QUIC en dispositivos *edge*, como en el caso de [28], se evalúa la reducción de latencia que se consigue al emplear QUIC. Se consideran diferentes escenarios IoT donde los equipos se envían mensajes MQTT sobre QUIC y TCP. Se mide el tiempo que tardan dichas comunicaciones y se compara el comportamiento de MQTT con ambos protocolos de transporte. Se contemplan canales libres de errores y otros

en los que se inyectan pérdidas con valores de Frame Error Rate (FER) realistas, sobre distintas tecnologías inalámbricas como WiFi, 4G/LTE y enlaces satelitales.

III. IMPLEMENTACIÓN

La implementación de QUIC que se ha utilizado en este trabajo se basa en la versión draft del IETF, y está desarrollada en el lenguaje de programación GO, *quic-go*[2]. El cliente y servidor de MQTT se basan en el código *open-source* de Eclipse Paho[3] y VolantMQ[4] respectivamente. Al igual que *quic-go*, estas aplicaciones también están desarrolladas en GO. Soportan la especificación al completo de MQTT, concretamente las versiones 3.1 y 3.1.1. Eclipse Paho implementa una librería que permite conectar el cliente MQTT con el broker VolantMQ usando TCP, TLS o *WebSocket*. En base a estas implementaciones del cliente y broker MQTT, se ha integrado QUIC en ambos proyectos. La Figura 2 muestra un resumen de los cambios realizados sobre las implementaciones originales (*net.go* y *quic_udp.go*), para habilitar el uso de MQTT sobre QUIC.

En el caso del cliente, todas las funcionalidades que se utilizan en la capa de transporte, abrir o cerrar la conexión o enviar y recibir paquetes, están gestionadas desde la interfaz *net.go*. Gracias a esto, la integración del socket de QUIC se agiliza, ya que solo requiere cambios mínimos en el código original para integrar conexiones basadas en TCP, TCP+TLS y WebSocket. En lo que se refiere a la integración de las conexiones QUIC para MQTT, la interfaz *net.go* llama a la interfaz *client.go*, implementada en *quic-go*. En dicha implementación, se soporta la funcionalidad de 0-RTT para el establecimiento de la conexión, a través de la función *DialAddrEarly* (*client.go*). Esta función, además de encargarse de abrir una sesión con el servidor, permite mandar datos antes de que el *handshake* de QUIC finalice reduciendo la latencia en el inicio de la conexión. Esta implementación de MQTT con QUIC es *open-source* y está disponible y accesible en un repositorio público[5].

La implementación *open-source* del *broker* con QUIC también está disponible en un repositorio git[6]. En este caso, las funciones de la capa de transporte están implementadas usando la interfaz *transport/conn.go*. La implementación original presenta algunas restricciones y, debido a la incompatibilidad entre las interfaces de conexión de TCP y QUIC, se ha decidido que esta implementación solo soporte el socket de QUIC. Por lo tanto, se integra la interfaz *quic_udp.go*, la cual llama al *listener* de la interfaz *server.go* de *quic-go*. La función que se utiliza para habilitar en el lado del servidor el 0-RTT es *ListenAddrEarly*. Esta función habilita que un cliente que se haya conectado al *broker* previamente pueda utilizar la información almacenada en la caché de esa sesión, y así en el restablecimiento de la conexión no tengan que negociar nuevamente todos los parámetros. De esta forma, el intercambio de datos sucede antes de que finalice el *handshake*.

La Figura 2 muestra un esquema de la comunicación entre cliente y *broker*. El *broker* estará escuchando en el socket que se le especifique. Se ejecutará la implementación original si se requiere TCP/TLS o la modificación realizada en el marco de este trabajo si se prefiere QUIC. En el caso de realizarse una conexión sobre QUIC, el servidor utilizará la función *ListenAddrEarly()* y el cliente establecerá la sesión a partir de *DialAddrEarly()*. De esta manera, ambos aceptarán utilizar el mecanismo de 0-RTT. Tras crearse la sesión, el cliente abrirá un *stream* a través de *OpenStreamSync()* y el *broker* la aceptará mediante la función *AcceptStream()*.

Por último, cabe destacar que se han identificado algunos aspectos que evidencian la falta de optimización de la implementación de QUIC. Los protocolos de transporte, TCP en particular, son capaces de combinar múltiples paquetes de protocolos de nivel superior en un único paquete de capa de transporte (*piggybacking*). Uno de los problemas que se han encontrado es la imposibilidad de analizar aquellos escenarios que conllevan una transmisión de datos a ráfagas de los *publishers*. Esto se debe a que la versión que se está utilizando en este trabajo de *quic-go* (v0.15.1) no es capaz de agrupar múltiples paquetes de MQTT aunque pertenezcan al mismo *stream* de datos.

IV. ENTORNO DE SIMULACIÓN

Como herramienta de evaluación, para estudiar el rendimiento de MQTT sobre QUIC, se ha empleado el simulador de redes basado en eventos discretos ns-3. En concreto, entre todas las funcionalidades de ns-3 se hace uso de la posibilidad de conectar aplicaciones en tiempo real sobre contenedores LXC de Linux, mediante una red simulada. Se ha generado un entorno de simulación basado en el escenario propuesto en la Figura 3, que muestra dos contenedores Linux conectados mediante ns-3. Esto permite emular varias tecnologías inalámbricas modelando dos parámetros principales: el ancho de banda y el retardo.

Un contenedor ejecuta la aplicación del cliente, compuesta por un *publisher* y un *subscriber*, mientras que el otro hace las veces de broker/servidor. ns-3 interpreta a los contenedores Linux como nodos fantasma conectados mediante una red CSMA a otro nodo, considerado como router. Para asegurar que el cuello de botella se encuentra en el enlace inalámbrico, ya que es el elemento de interés en el estudio, se configura una capacidad alta (10Gbps) en los enlaces entre los nodos y los contenedores. Los routers están conectados mediante un enlace punto a punto (P2P), que se ajusta en función del ancho de banda, retardo y la tasa de pérdidas.

La Tabla I muestra los parámetros empleados para modelar las diferentes redes propuestas. Los buffers han

[2]Implementación de QUIC https://github.com/lucas-clemente/quic-go versión v0.15.1.

[3]Cliente MQTT Eclipse Paho, https://github.com/eclipse/paho.mqtt.golang versión v1.2.0.

[4]Broker MQTT, https://github.com/VolantMQ/volantmq versión v0.4.0-rc6.

[5]Cliente Eclipse Paho MQTT con el socket de QUIC, https://github.com/pgOrtiz90/paho.mqtt.golang

[6]Servidor/*broker* MQTT con soporte de QUIC, https://github.com/fatimafp95/volantmq_2

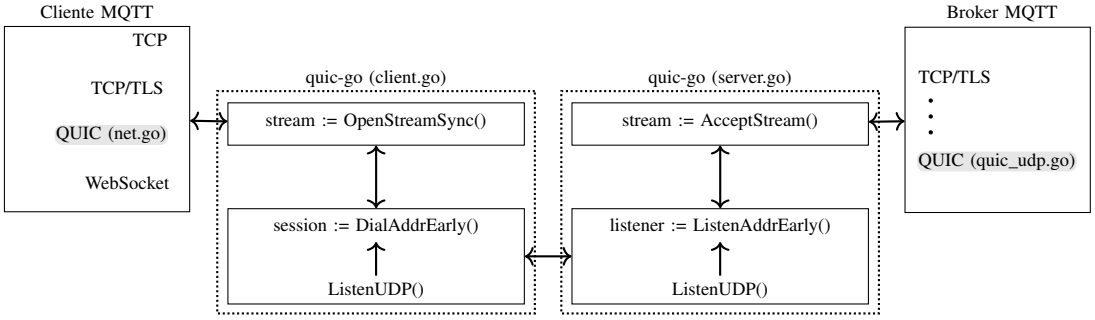


Fig. 2. Esquema de la integración de QUIC como protocolo de transporte para servicios basados en MQTT.

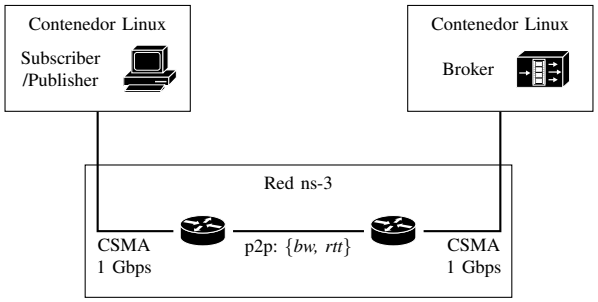


Fig. 3. Escenario inicial con dos contenedores Linux conectados por una red P2P: el contenedor de la izquierda ejecuta el cliente MQTT y el de la derecha el broker.

Tabla I
PARÁMETROS DE RED PARA EMULAR DISTINTAS TECNOLOGÍAS.

	Tipo1	**Tipo2**	**Tipo3**
	WiFi	**4G/LTE**	**Satélite**
Capacidad [Mbps]	20	10	1.5
RTT [ms]	25	100	600
Tasa de pérdidas [%]	[0, 1, 2, 3, 5, 10]		

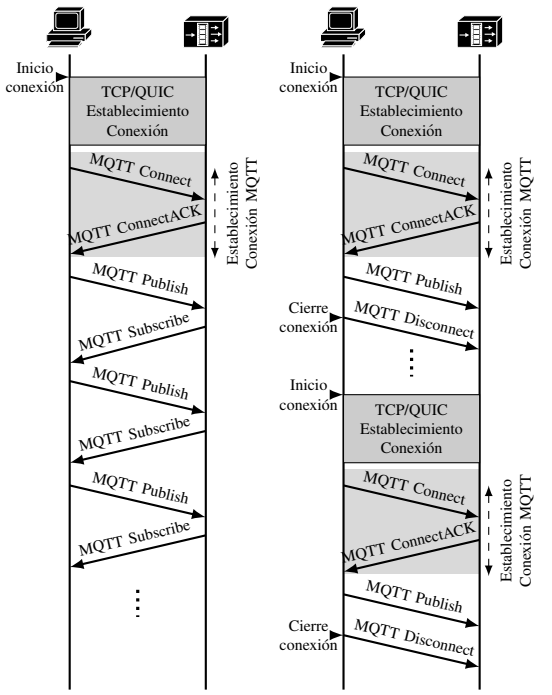
sido configurados a un Bandwidth Delay Product (BDP), teniendo aunque hemos tenido en consideración el denominado *bufferbloat effect* [29]. Este efecto se produce al tener tamaños de buffer superiores al BDP, algo característico en muchas redes móviles.

Por otro lado, ns-3 permite el uso de redes WiFi para conectar contenedores Linux, permitiendo analizar el rendimiento conjunto de MQTT y QUIC sobre un canal compartido. ns-3 proporciona herramientas para configurar la capa MAC y física, permitiendo fijar así el modelo de tasa de error. La interfaz que nos permite controlar este modelo de tasa de error en Wi-Fi calcula la probabilidad de recibir correctamente los paquetes sobre la capa física. Sin embargo, para emular las mismas condiciones del primer escenario, se ha modificado dicha interfaz, permitiendo controlar la tasa de pérdidas mostrada en la Tabla I, independientemente del modelo de interferencias empleado.

La configuración realizada está basada en una red inalámbrica ad-hoc. Dicha red conecta varios contenedores Linux como *publishers* MQTT al *broker*, el cual a su vez está conectado a un *subscriber* a través de una red similar.

V. RESULTADOS

En este trabajo se ha extendido la experimentación descrita en [30] para obtener un análisis más completo del comportamiento de MQTT sobre QUIC.



(a) Escenario con una única conexión MQTT.

(b) Escenario con múltiples conexiones MQTT.

Fig. 4. Escenarios implementados en la fase de experimentación.

En el primer escenario, que se muestra en la Figura 4a, se mandan 100 mensajes MQTT desde el *publisher* hasta el *broker*, el cual reenvía el mensaje correspondiente al subscriber. Se ejecuta cada experimento 50 veces para garantizar la validez estadística de los resultados. Se establece una sola conexión MQTT y no se cierra hasta que termine la transmisión de todos los mensajes. En este caso, el servicio que ofrece MQTT sigue un comportamiento *stop&wait*, donde el paquete i-*ésimo* solo se envía después de recibir la suscripción del anterior. Se analiza el tiempo necesario para transmitir todos los paquetes con TCP y QUIC, T_{TCP} y T_{QUIC} respectivamente. Para evaluar la reducción de T_{QUIC} frente a T_{TCP}, se define el parámetro ratio de finalización ξ como se muestra en la ec. 1

$$\xi = \frac{T_{\text{QUIC}}}{T_{\text{TCP}}} \tag{1}$$

Así, cuando $\xi < 1$, se puede decir que QUIC supera a TCP, ya que el tiempo que se alcanza para transmitir toda la información entre los endpoints es menor que el que se

obtiene con TCP. Este parámetro se mide sobre las tres redes que se configuran con los parámetros descritos en la Tabla I, con distintas tasas de pérdida de paquetes.

Para realizar un análisis más extenso, se profundiza en la metodología y configuración de [30] para generar escenarios más complejos. Considerando una arquitectura fog/cloud para IIoT y la configuración de la Figura 3, se añade otro nodo para separar los roles de los clientes de MQTT, donde el *publisher* y el *subscriber* se conectan al *broker* a través de distintas redes.

La Figura 5 muestra el ratio de finalización separando los clientes en distintos contenedores LXC. QUIC, en línea a lo analizado en [30], sigue presentando una menor latencia que TCP durante el intercambio de datos, especialmente sobre redes con RTT pequeños y tasa de pérdida alta gracias a sus mecanismos de recuperación de paquetes. Además, se deduce que cuando existen RTT elevados, la mejora de QUIC puede ser menos notable, como se puede ver en los enlaces satelitales. Sin embargo, QUIC sigue presentando una reducción de la latencia de aproximadamente 35% sobre redes Wi-Fi y un 5% de pérdidas.

Uno de los puntos fuertes del diseño de QUIC es la reducción temporal en el restablecimiento de la conexión. Para analizar esta mejora, y compararla con el esquema típico de TCP/TLS, se ejecuta el escenario 4b. Se cierra la conexión después de que el *publisher* mande un mensaje a un *topic* específico.

En [30] se muestra que aunque QUIC consiga mejorar las prestaciones de TCP, no aprovecha el 0-RTT. Después de analizar la implementación de *quic-go* se comprobó que había un problema con la gestión de los paquetes 0-RTT, que perjudicaba a QUIC. Esto se debía a que la implementación del servidor de QUIC no procesaba los paquetes 0-RTT antes de que el cliente retransmitiera datos 0-RTT en los paquetes 1-RTT. Después de detectar este inconveniente, los resultados mostrados en la Figura 6 avalan que QUIC maneja tiempos más cortos en el establecimiento de la conexión que el esquema tradicional TCP/TLS, concretamente cuando los nodos vuelven a comunicarse. Se ha ejecutado 50 veces la configuración de la Figura 3, separando los roles del cliente MQTT. Se han establecido dos conexiones para probar el impacto en la reconexión y se mide el tiempo desde que se establece la conexión inicial del *publisher* hasta que el *subscriber* recibe el segundo mensaje. En todos los casos QUIC supera en rendimiento a TCP/TLS, especialmente en enlaces con un RTT alto como los habituales en redes satelitales.

Teniendo en cuenta que el simulador ns-3 permite utilizar redes WiFi, así como los diferentes intercambios de tramas identificados en la Figura 4, se han utilizado varios clientes MQTT conectados a un *broker* mediante una red WiFi. Este escenario podría emular una red inalámbrica de sensores que se conectan a una arquitectura fog/cloud, en la que varios dispositivos edge generan datos que serán procesados por la capa fog. Finalmente, en el caso de ser necesario, serán enviados al cloud (rol del *subscriber*) a través de un enlace cableado con bajo RTT (25 ms) y un elevado ancho de banda.

La Figura 7 representa el parámetro ξ en el escenario descrito en 4a, al modificar la tasa de error y el número de retransmisiones que realiza la capa MAC. El experimento se ha realizado 50 veces para cada tasa de error y número de retransmisiones MAC. Con el objetivo de reducir el tiempo de simulación, se transmiten 100 paquetes MQTT desde el *publisher* al *subscriber*. La Figura 7 muestra que QUIC mejora el comportamiento de TCP, por lo que podría ayudar la latencia en redes WiFi y, en general, inalámbricas.

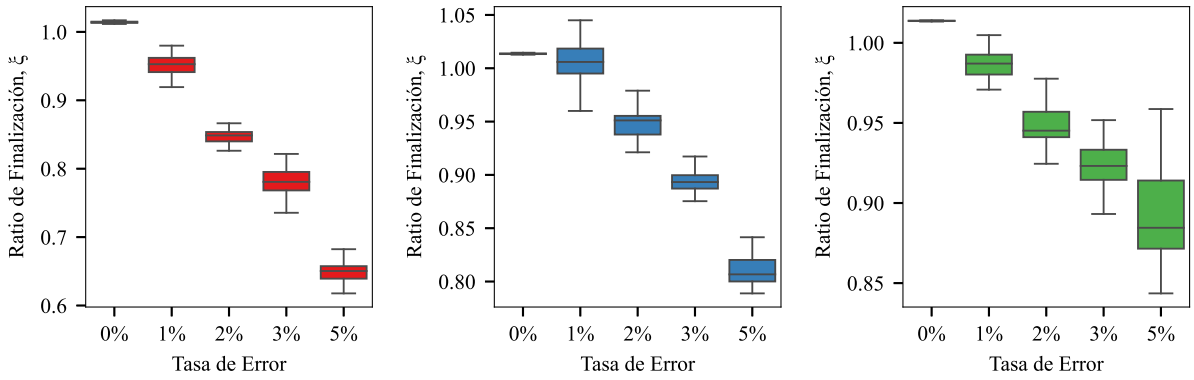
Por otro lado, se ha ejecutado un *subscriber* suscrito a un *topic* especifico variando el número de *publishers* en tres, cinco y ocho, donde uno de ellos publica información en dicho *topic*. La Figura 8 muestra el rendimiento en canales compartidos sobre el escenario descrito en la Figura 4a. QUIC mejora el rendimiento de TCP/TLS para todos los casos, mostrando un comportamiento más estable (con menor variabilidad) a medida que se incluyen más *publishers* generando datos.

VI. CONCLUSIONES Y LÍNEAS FUTURAS

En este paper se ha discutido y evaluado la alternativa de emplear QUIC como protocolo de transporte frente a la combinación TCP/TLS en aplicaciones industriales IIoT para reducir el retardo en estos entornos. En este tipo de escenarios la latencia es uno de los parámetros más críticos y, por tanto, este trabajo la analiza para comparar ambos protocolos. Las dos principales contribuciones de este paper son: en primer lugar la implementación basada en GO de MQTT sobre QUIC, y en segundo lugar, el análisis del rendimiento de dicha implementación en los escenarios IIoT emulados.

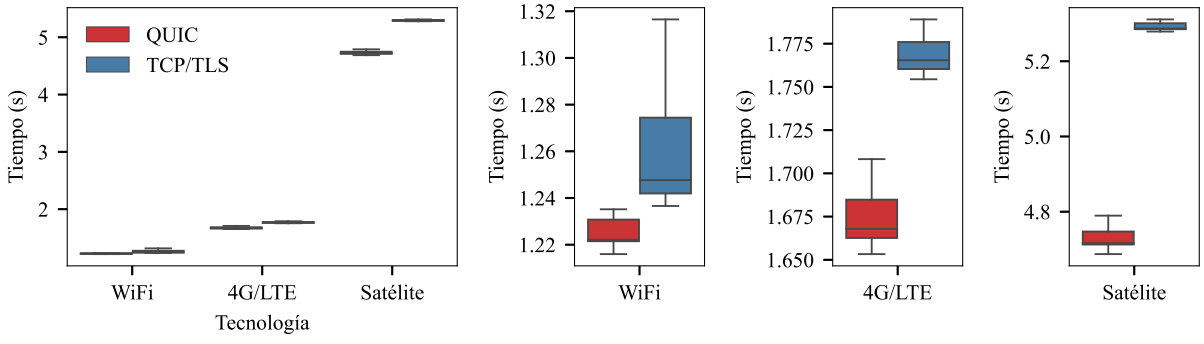
Por un lado, se han configurado tres tipos de redes en el simulador ns-3: Wi-Fi, 4G y satelital. Para ello se ha ajustado el ancho de banda y el retardo para emular diferentes tecnologías. Por otro lado, ns-3 permite también emular un escenario más realista, en el que clientes MQTT (things) publican mensajes a un *broker* (fog) haciendo uso de una red Wi-Fi. El *broker* a su vez transmite la información a un subscriber (cloud) conectado a través de una red P2P que emularía la conexión entre fog y cloud. Gracias a este simulador, y los contenedores Linux, se ha evaluado la combinación de QUIC y MQTT, comparando su comportamiento frente al ofrecido por la solución tradicional de MQTT/TLS/TCP.

A pesar de las restricciones derivadas de las implementaciones, se han diseñado dos escenarios complementarios que permiten analizar MQTT con QUIC y TCP. Por un lado, con el primer escenario se ha comprobado que QUIC excede en rendimiento a TCP en el intercambio de mensajes entre los nodos, especialmente sobre conexiones con un bajo RTT, volviéndose más evidente a medida que incrementa el número de paquetes que se pierden en la red. Se ha comprobado que el impacto del ancho de banda es casi insignificante, ya que se consideran paquetes de pequeño tamaño, típicos en IoT. Por otro lado, el segundo



(a) WiFi (b) 4G/LTE (c) Satélite

Fig. 5. Comportamiento de MQTT sobre QUIC y TCP para el escenario 4a.



(a) (b) (c) (d)

Fig. 6. 0-RTT con tres nodos: *publisher-broker-subscriber*. 6a muestra las tres configuraciones, y 6b, 6c y 6d el tiempo de transmisión para cada red.

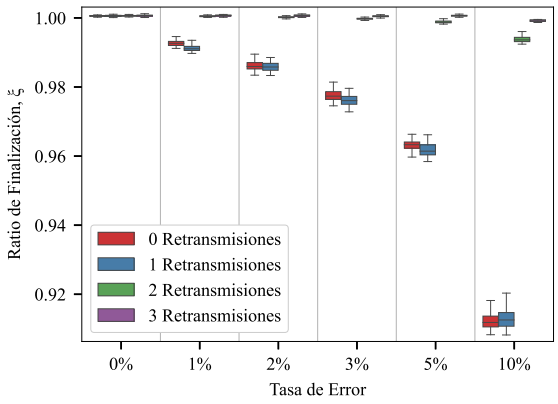


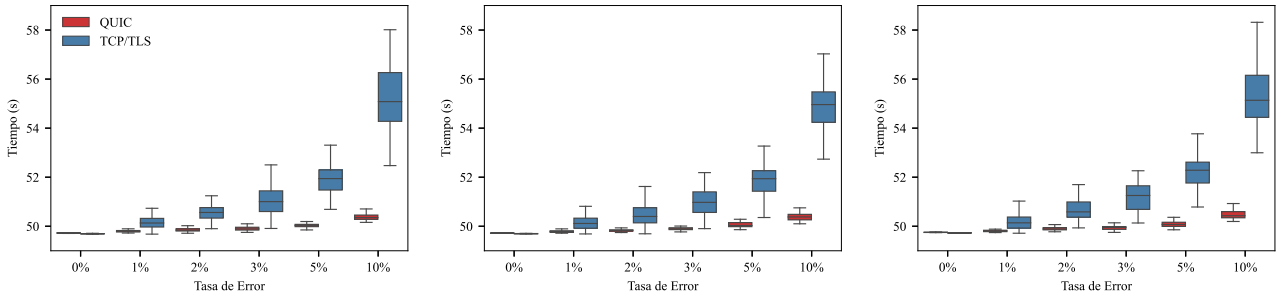
Fig. 7. *Boxplot* con el número de retransmisiones WiFi.

escenario fue concebido para conocer los beneficios del esquema 0-RTT que QUIC promueve. Los resultados muestran una clara reducción en la latencia durante el establecimiento de la conexión. Este logro podría ayudar a alcanzar comunicaciones más fiables en redes inalámbricas de entornos industriales. Por último, se ha evaluado QUIC en canales compartidos, configurando una red Wi-Fi en el simulador ns-3. Además, se ha concluido que QUIC presenta un comportamiento aceptable sobre entornos compartidos, como son las redes de sensores. Todos estos resultados se llevan a cabo para evaluar el rendimiento de QUIC frente a TCP en términos de latencia. QUIC ofrece tiempos más cortos, debido a los mecanismos de recuperación de pérdidas y a la funcionalidad 0-RTT.

Gracias al prematuro estado de desarrollo de QUIC se han encontrado aspectos interesantes que pueden ser aplicados, como añadir técnicas multipath en la implementación de *quic-go* o estudiar mecanismos de control para la congestión en IoT. Para garantizar la estabilidad y la disponibilidad de la red, es esencial evaluar los eventos de congestión e intentar evitarlos. Combinar multipath en QUIC junto con el algoritmo de control de congestión podría ser interesante, ya que los eventos de congestión se reducirían significativamente. La conexión multipath podría aliviar los cuellos de botella que se generan, redirigiendo el tráfico por rutas con menos carga. En este trabajo se considera que estas propuestas constituyen interesantes puntos de estudio para su uso en el entorno IIoT en los que la fiabilidad, la latencia y la capacidad son de vital importancia.

AGRADECIMIENTOS

Los autores agradecen la financiación del Programa de Doctorados Industriales de la Universidad de Cantabria (convocatoria 2020). El trabajo ha sido financiado por el Gobierno Vasco a través del programa Elkartek, y el proyecto DIGITAL (KK-2019/0009), y por la Agencia Estatal de Investigación, proyecto FIERCE: Future Internet Enabled Resilient smart CitiEs (RTI2018-093475-AI00).



(a) 3 *publishers* (b) 5 *publishers* (c) 8 *publishers*

Fig. 8. Comportamiento de MQTT sobre QUIC y TCP/TLS en canales compartidos con el escenario 4a.

REFERENCIAS

[1] M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.

[2] G. Aceto, V. Persico, and A. Pescapé, "A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies, Perspectives, and Challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3467–3501, 2019.

[3] A. Varghese and D. Tandur, "Wireless requirements and challenges in Industry 4.0," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, nov 2014, pp. 634–638.

[4] A. Banks and R. Gupta, "MQTT version 3.1.1," International Organization for Standardization (ISO), Standard, 2014.

[5] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT version 5.0," Organization for the Advancement of Structured Information Standards (OASIS), Standard, 2019.

[6] J. Postel, "Transmission control protocol," Internet Requests for Comments, RFC Editor, STD 7, September 1981, http://www.rfc-editor.org/rfc/rfc793.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[7] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in iot networking via tcp/ip architecture," *NDN Project*, 2016.

[8] Chonggang Wang, K. Sohraby, Bo Li, M. Daneshmand, and Yueming Hu, "A survey of transport protocols for wireless sensor networks," *IEEE Network*, vol. 20, no. 3, pp. 34–40, may 2006.

[9] J. Luo, J. Jin, and F. Shan, "Standardization of Low-Latency TCP with Explicit Congestion Notification: A Survey," *IEEE Internet Computing*, vol. 21, no. 1, pp. 48–55, 2017.

[10] R. Stewart, "Stream control transmission protocol," Internet Requests for Comments, RFC Editor, RFC 4960, September 2007, http://www.rfc-editor.org/rfc/rfc4960.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4960.txt

[11] C. Zhang and V. Tsaoussidis, "Tcp-real: Improving real-time capabilities of tcp over heterogeneous networks," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 189–198.

[12] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Medard, "Network Coded TCP (CTCP)," *arXiv e-prints*, p. arXiv:1212.2291, 2012.

[13] A. Langley, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, A. Riddoch, W.-t. Chang, Z. Shi, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, and I. Swett, "The QUIC Transport Protocol," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*. New York, New York, USA: ACM Press, 2017, pp. 183–196.

[14] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, IETF, Tech. Rep. 9000, May 2021, https://rfc-editor.org/rfc/rfc9000.txt. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[15] ——, "Quic: A udp-based multiplexed and secure transport," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-transport-27, February 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt

[16] M. Thomson and S. Turner, "Using tls to secure quic," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-tls-27, February 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-27.txt

[17] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008, http://www.rfc-editor.org/rfc/rfc5246.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[18] E. Rescorla, "The transport layer security (tls) protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018.

[19] J. Iyengar and I. Swett, "Quic loss detection and congestion control," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-quic-recovery-27, March 2020, http://www.ietf.org/internet-drafts/draft-ietf-quic-recovery-27.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-quic-recovery-27.txt

[20] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 181–194, 2011.

[21] P. Wang, C. Bianco, J. Riihijärvi, and M. Petrova, "Implementation and performance evaluation of the quic protocol in linux kernel," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 227–234.

[22] Q. De Coninck, F. Michel, M. Piraux, F. Rochet, T. Given-Wilson, A. Legay, O. Pereira, and O. Bonaventure, "Pluginizing quic," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 59–74.

[23] A. Čolaković and M. Hadžialić, "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, pp. 17–39, oct 2018.

[24] Nikshepa and V. Pai, "Survey on iot security issues and security protocols," *International Journal of Computer Applications*, vol. 180, no. 42, pp. 16–21, May 2018.

[25] E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. Ramakrishnan, "Robustness of IoT Application Protocols to Network Impairments," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, jun 2018, pp. 97–103.

[26] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[27] P. Kumar and B. Dezfouli, "Implementation and analysis of QUIC for MQTT," *Computer Networks*, vol. 150, pp. 28–45, feb 2019.

[28] L. Eggert, "Towards securing the internet of things with quic," EasyChair Preprint no. 2434, EasyChair, 2020.

[29] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling bufferbloat in 3G/4G networks," in *Proc. of the 2012 ACM conference on Internet Measurement Conference (IMC)*, 2012, pp. 329–342.

[30] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, and R. Agüero, "And quic meets iot: performance assessment of mqtt over quic," in *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2020, pp. 1–6.