UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA

MAESTRÍA EN INGENIERÍA EN REDES DE COMUNICACIÓN

"DESARROLLO DE UNA TÉCNICA DE EVALUACIÓN DE DESEMPEÑO DE PROTOCOLOS PARA LA CAPA DE APLICACIÓN UTILIZADOS EN LA INTERNET DE LAS COSAS"

Tesis de Grado

Presentado para obtener el Título de Magister en Ingeniería de Redes de Comunicación

Postulante: Ing. Jorge Hernan Ticona Sanga

Tutor: MSc. Roberto Guido Zambrana Flores

La Paz – Bolivia

2023

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERIA

LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) Visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) Copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) Copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la cita o referencia correspondiente en apego a las normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
INSTITUTO DE ELECTRÓNICA APLICADA

DESARROLLO DE UNA TÉCNICA DE EVALUACIÓN DE DESEMPEÑO DE PROTOCOLOS PARA LA CAPA DE APLICACIÓN UTILIZADOS EN LA INTERNET DE LAS COSAS

Jorge Hernan Ticona Sanga

Tesis de maestría presentada para el Curso de Maestría en Ingeniería en Redes de Comunicación 8va. Versión, orientado por el MSc. Roberto Guido Zambrana Flores, aprobada en 08 de agosto por extenso de 2023.

IEA - UMSA
La Paz
2023

A mi familia, a mi tutor, a mis amigos, tanto aquellos que están a mi lado físicamente como aquellos que están conmigo en la distancia a través de internet, y a todas las personas que persiguen sus sueños a pesar de las dificultades.

“I'm not interested in a good man's life. I'm interested in contradiction.”

Cillian Murphy.

RESUMEN

El Internet de las Cosas, IoT, encierra la idea de un mundo compuesto por objetos que contienen tecnología embebida que pueden interconectarse a través de internet. Estos objetos, en general, tienen limitado el procesamiento, almacenamiento, potencia de transmisión y fuente de energía, por lo que se crearon o adaptaron protocolos que cumplan estas necesidades para las diferentes capas que componen el modelo Open System Interconnection (OSI).

Los protocolos de la capa de aplicación del modelo OSI se caracterizan por admitir el intercambio de datos entre los programas que se ejecutan en los hosts de origen y destino. Los principales protocolos utilizados en esta capa para IoT son: Protocolo de Transferencia de Hipertexto Transferencia de Estado Representacional (HTTP/2), Protocolo de Mensajería Avanzado en Cola (AMQP), Protocolo de Aplicación Restringido (CoAP), y Protocolo Cola de Mensaje de Telemetría y Transporte (MQTT).

En el presente trabajo se presenta una técnica de evaluación de desempeño de protocolos para la capa de aplicación utilizados en la Internet de las Cosas (IoT), que permite medir y comparar el rendimiento de estos protocolos en términos de latencia, pérdida de paquetes y rendimiento (throughput). La técnica se basa en el diseño e implementación de pruebas experimentales, la recolección y análisis de datos, y la definición de recomendaciones prácticas para la selección adecuada de protocolos.

La técnica se aplica a dos protocolos usados en la capa de aplicación para IoT: AMQP y MQTT. Los resultados obtenidos muestran las ventajas y desventajas de cada protocolo en diferentes escenarios y condiciones, y proporcionan una base sólida para la toma de decisiones informadas y el diseño de sistemas IoT más eficientes.

Palabras clave: IoT. HTTP/2. CoAP. AMQP. MQTT. Estadística descriptiva.

ABSTRACT

The Internet of Things, IoT, contains the idea of a world made up of objects that contain embedded technology that can be interconnected through the Internet. These objects, in general, have limited processing, storage, transmission power and energy source, so protocols were created or adapted to meet these needs for the different layers that make up the Open System Interconnection (OSI) model.

The application layer protocols of the OSI model are characterized by supporting the exchange of data between programs running on the source and destination hosts. The main protocols used at this layer for IoT are: Hypertext Transfer Protocol Representational State Transfer (HTTP/2), Advanced Messaging Queuing Protocol (AMQP), Restricted Application Protocol (CoAP), and Message Queuing Protocol. Telemetry and Transport (MQTT).

This paper presents a technique for evaluating the performance of protocols for the application layer used in the Internet of Things (IoT), which allows measuring and comparing the performance of these protocols in terms of latency, packet loss and throughput. (throughput). The technique is based on the design and implementation of experimental tests, the collection and analysis of data, and the definition of practical recommendations for the appropriate selection of protocols.

The technique is applied to two protocols used in the application layer for IoT: AMQP and MQTT. The results obtained show the advantages and disadvantages of each protocol in different scenarios and conditions, and provide a solid foundation for making informed decisions and designing more efficient IoT systems.

Keywords: IoT. HTTP/2. CoAP. AMQP. MQTT. Descriptive statistics.

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE SIGLAS Y ABREVIATURAS

IoT	Internet de las Cosas
OSI	Interconexión de Sistemas Abiertos
HTTP	Protocolo de Transferencia de Hipertexto Transferencia de Estado Representacional
AMQP	Protocolo de Mensajería Avanzado en Cola
CoAP	Protocolo de Aplicación Restringido
MQTT	Protocolo Cola de Mensaje de Telemetría y Transporte
KSTAR	Korea Superconducting Tokamak Advanced Research
QoS	Calidad de Servicio

SUMARIO

1. INTRODUCCIÓN

1.1. Antecedentes

La Internet de las Cosas, IoT, se refiere a la red colectiva de dispositivos conectados a través de una tecnología, que además permite la comunicación de estos dispositivos con la nube. Los dispositivos que conforman esta red se caracterizan por usar chips de procesamiento de bajo coste. En la actualidad podemos decir que existen miles de millones de dispositivos conectados a internet, de los cuales pueden ser cepillos de dientes, aspiradoras y coches entre muchos otros objetos de uso común, que ahora ya cuentan con sensores para recopilar datos y responder en forma inteligente a los usuarios (¿Qué es IoT?, s.f.).

La Internet de las Cosas conecta dispositivos de uso diario a una red, que por lo general es internet. El desarrollo tecnológico en este sector fue lento, se puede decir que se comenzaron a implementar sensores y procesadores a los objetos cotidianos desde los años 90, aunque los chips no eran pequeños. Las etiquetas RFID, que son chips de ordenador de baja frecuencia, se utilizaron primeramente para el seguimiento de dispositivos caros, con la reducción de tamaño de los chips, estos se hacían más rápidos y útiles para otros objetos que no contaban con chips ni con conectividad a internet (¿Qué es IoT?, s.f.).

El coste de la integración de chips con buena potencia de procesamiento en objetos pequeños se redujo en gran medida, lo cual ha permitido el desarrollo del sector con la meta de que en los hogares y empresas se llenen de objetos IoT. Estos objetos inteligentes pueden transmitir automáticamente datos a internet. Cabe recalcar que estos objetos y la tecnología asociada a ellos se denominan de manera conjunta como internet de las cosas (¿Qué es IoT?, s.f.).

En este escenario los dispositivos IoT al conectarse entre sí y la nube, generan grandes cantidades de datos que se transmiten y procesan a través de modelo OSI (Open Systems Interconnection), la capa de aplicación que se encuentra en la parte superior del modelo OSI permite la interacción y el intercambio de información entre los dispositivos IoT, a medida que el ecosistema IoT crece, surgen retos en cuanto a la selección y el desempeño de los protocolos que funcionan en la capa de aplicación.

La elección de protocolos para la capa de aplicación utilizados en la IoT es importante para garantizar el funcionamiento óptimo de los sistemas. Por lo que es recomendable contar con técnicas de evaluación de desempeño sólidas y confiables que permitan analizar y comparar el desempeño de los protocolos en diferentes escenarios y condiciones, identificando métricas de desempeño adecuadas, definiendo métodos de evaluación y ejecutando los experimentos.

Al enfrentarnos a este reto, se espera obtener beneficios importantes para el desarrollo y la implementación de sistemas IoT, una técnica de evaluación de desempeño de protocolos para la capa de aplicación permitirá a los profesionales de la industria IoT seleccionar y optimizar los protocolos más adecuados para sus aplicaciones específicas, mejorando de esta manera la eficiencia de la comunicación y la calidad general de los sistemas IoT.

1.2. Planteamiento del problema

En el contexto en el que se viene desarrollando la IoT, la interconexión de dispositivos a través de redes de comunicación ha experimentado un rápido crecimiento en los últimos años. Este desarrollo tecnológico ha transformado diversos rubros como pueden ser la industria, salud y hogar, permitiendo de esta forma mayor automatización y optimización de procesos. Pero a medida que el ecosistema IoT crece, surgen desafíos relacionados con los protocolos utilizados en la capa de aplicación, que son responsables de habilitar la interacción de datos entre los dispositivos IoT.

Se debe seleccionar adecuadamente los protocolos para la capa de aplicación utilizados en la IoT, debido a la variedad de protocolos disponibles, resulta difícil determinar cuál es el más apropiado con respecto desempeño. La selección incorrecta puede conducir a incompatibilidades, ineficiencias y limitaciones en el funcionamiento de los dispositivos IoT, afectando negativamente el rendimiento de los sistemas. Para eso, es importante la evaluación y comparación del desempeño de los protocolos utilizados en la capa de aplicación. La eficiencia de los protocolos es fundamental para garantizar una comunicación efectiva y confiable en la IoT.

Por lo tanto, es necesario desarrollar una técnica de evaluación de desempeño de protocolos para la capa de aplicación en la IoT. Esta técnica permitirá una selección adecuada de los protocolos de aplicación, considerando los aspectos de rendimiento y las restricciones de recursos de los dispositivos IoT. Esta debe proporcionar debe proporcionar una evaluación completa y precisa del desempeño de los protocolos, utilizando métricas relevantes y tomando en cuenta los requisitos de la capa de aplicación en la IoT.

Se espera mejorar la eficiencia, el rendimiento y la confiabilidad de los sistemas IoT, lo que resultará en un uso más eficiente de los recursos y una mejor calidad de servicio. Además, esta técnica proporciona una base sólida para la selección de los protocolos de aplicación utilizados en la IoT, facilitando a los profesionales de este rubro la toma de decisiones informadas y el diseño de sistemas IoT más eficientes.

1.3. Objetivo

Desarrollar una técnica de evaluación de protocolos para la capa de aplicación utilizados en la IoT como estrategia para medir su desempeño.

1.4. Objetivos específicos

- Documentar el estado del arte de los protocolos para la capa de aplicación utilizados en la IoT para entender sus características y especificaciones.
- Diseñar e implementar una técnica de evaluación para estos protocolos con la finalidad de medir su desempeño.
- Proponer recomendaciones para la selección de protocolos en base a los resultados obtenidos de la técnica de evaluación.

1.5. Hipótesis y operacionalización de variables de la investigación

1.5.1. Hipótesis

Al desarrollar una técnica de evaluación de desempeño de protocolos para la capa de aplicación utilizados en la IoT permitirá mejorar significativamente la eficiencia de los sistemas IoT.

Brindando recomendaciones basadas en evidencias para la selección de protocolos adecuados, se logrará una optimización integral de los dispositivos y aplicaciones IoT mejorando la calidad de servicio y la experiencia del usuario final.

1.5.2. Variable dependiente

Desempeño de Protocolos de Aplicación en la Internet de las Cosas.

1.5.3. Variable independiente

Protocolos para la Capa de Aplicación utilizados en la Internet de las Cosas.

1.5.4. Variable interviniente

Técnica de Evaluación de Desempeño de Protocolos para la Capa de Aplicación utilizados en la Internet de las Cosas.

1.6. Operacionalización de variables

Las siguientes tablas a continuación muestran a continuación la operacionalización de las variables.

Tabla 1. Operacionalización de la variable independiente

Variable Independiente	Definición Conceptual	Definición Operacional	Componentes	Dimensión	Indicadores
Protocolos para la Capa de Aplicación utilizados en la Internet de las Cosas.	Protocolos utilizados para suministrar servicios de red a las aplicaciones del usuario.	Se realizará mediante el estudio de los protocolos.	Se identificarán los protocolos. Se recopilarán las características de los protocolos. Se seleccionarán los protocolos.	Características de los protocolos.	Tipo de arquitectura. Formato de mensaje.

Fuente: Elaboración propia.

Variable Dependiente: Desempeño de Protocolos de Aplicación en la Internet de las Cosas.

Tabla 2. Operacionalización de la variable dependiente

Variable Dependiente	Definición Conceptual	Definición Operacional	Componentes	Dimensión	Indicadores
Desempeño de Protocolos para la Capa de Aplicación utilizados en la Internet de las Cosas.	La medida de desempeño de los protocolos se hará en términos de latencia, pérdida de paquetes y rendimiento (*throughput*).	Especificación clara y precisa de cómo se medirán y cuantificarán las diferentes dimensiones del desempeño de los protocolos.	Pruebas experimentales. Recopilación de resultados. Análisis de resultados. .	Desempeño de los protocolos en términos de latencia, perdida de paquetes y rendimiento (*throughput*).	Resultado de las pruebas de: Latencia (ms). Perdida de paquetes (%). Rendimiento (*throughput*) (MB/s)

Fuente: Elaboración propia.

1.7. Justificación

La Internet de las Cosas se ha convertido en una de las áreas de desarrollo tecnológico de gran importancia en la actualidad, con aplicaciones en diferentes sectores tales como la industria, la salud y el hogar. A pesar de eso, a medida que IoT sigue en crecimiento, aparecen retos relacionados con los protocolos para la capa de aplicación utilizados en la IoT, los cuales son importantes para el funcionamiento eficiente y confiable de los sistemas IoT.

El presente trabajo se justifica en la necesidad de abordar estos retos y proporcionar una técnica de evaluación de desempeño adecuada. Al desarrollarse esta técnica, se logrará una selección más fundamentada de los protocolos. De esta manera se contribuirá a mejorar la eficiencia y el rendimiento de los sistemas IoT, ya que se elegirá a los protocolos más eficientes y adaptados a las necesidades específicas de cada aplicación.

Se tendrá un efecto positivo en rubros como la automatización industrial, la atención médica y la gestión de ciudades inteligentes, entre muchos otros. Debido a que al mejorar la eficiencia y el rendimiento de los sistemas IoT, esta investigación contribuirá al avance de estos sectores, de tal manera que se impulse la mejora de calidad de los servicios ofrecidos.

1.8. Alcance

En base a los objetivos planteados previamente, a continuación, se establecerán los alcances del presente trabajo.

En primer lugar, se documentará el estado del arte de los protocolos utilizados en la capa de aplicación de la IoT. Esto permitirá identificar y comprender las características relevantes de estos protocolos que serán fundamentales en el desarrollo de la técnica propuesta.

A continuación, se diseñará la técnica de evaluación de desempeño, teniendo en cuenta las características específicas de los protocolos y sus requisitos. La técnica deberá ser capaz de medir y analizar el desempeño de los protocolos de manera precisa y efectiva.

Una vez diseñada la técnica, se procederá a su implementación. Se realizarán pruebas experimentales utilizando escenarios representativos de aplicaciones IoT, para evaluar el desempeño de los protocolos, se hará uso de la estadística descriptiva como herramienta de análisis de los datos obtenidos en las pruebas. Posteriormente, se estudiarán los resultados obtenidos del análisis estadístico, con el objetivo de obtener conclusiones sobre el desempeño de los protocolos en términos de latencia, pérdida de paquetes y rendimiento (throughput).

Finalmente, a partir de los resultados y análisis obtenidos, se propondrán recomendaciones prácticas y orientativas para la selección adecuada de protocolos en la Capa de Aplicación de la IoT. Estas recomendaciones serán fundamentales para que los profesionales del campo puedan tomar decisiones informadas y diseñar sistemas IoT más eficientes.

Es importante mencionar que la investigación se enfocará exclusivamente en la evaluación de desempeño de los protocolos en la Capa de Aplicación, sin abordar aspectos relacionados con la seguridad de los protocolos o su interoperabilidad.

2. REVISIÓN DE LA LITERATURA

2.1. Internet de las cosas

El termino Internet de las Cosas fue acuñado por Peter T. Lewis en un discurso publicado en septiembre de 1985, en ese discurso Lewis indicó lo siguiente:

Internet de las cosas, o IoT, es la integración de personas, procesos y tecnología con dispositivos y sensores conectables para permitir el monitoreo, el estado, la manipulación y la evaluación remotos de las tendencias de dichos dispositivos (Internet of things, 2021).

Sin embargo, el concepto de Internet de las Cosas ha ido evolucionando con el pasar del tiempo y depende del autor. A continuación, se muestra la última definición que indica Gartner (s.f.-a):

Internet de las cosas (IoT) es la red de objetos físicos que contienen tecnología embebida para comunicarse y detectar o interactuar con sus estados internos o el entorno externo.

Esta definición indica que los objetos físicos IoT contienen tecnologías embebidas tales como: sensores, procesadores y software. Estos objetos tienen la capacidad de comunicarse, esto podría ser a través de una red local o internet, pueden detectar las condiciones externas a través de sus sensores y ejecutar acciones a través de actuadores según defina el software implementado.

Por lo general los dispositivos IoT se caracterizan por ser de tipo restringido. Sus capacidades de procesamiento y almacenamiento de información son limitados, cuentan con poca potencia de transmisión de datos y sus fuentes de energía son de poca capacidad y limitadas, esto se debe a que los dispositivos IoT deben ocupar pequeños espacios.

2.2. Patrones de comunicación utilizados en la IoT

Los patrones de comunicación identifican flujos comunes de mensaje que se usan en soluciones de mensajería (Messaging patterns, s.f.-b).

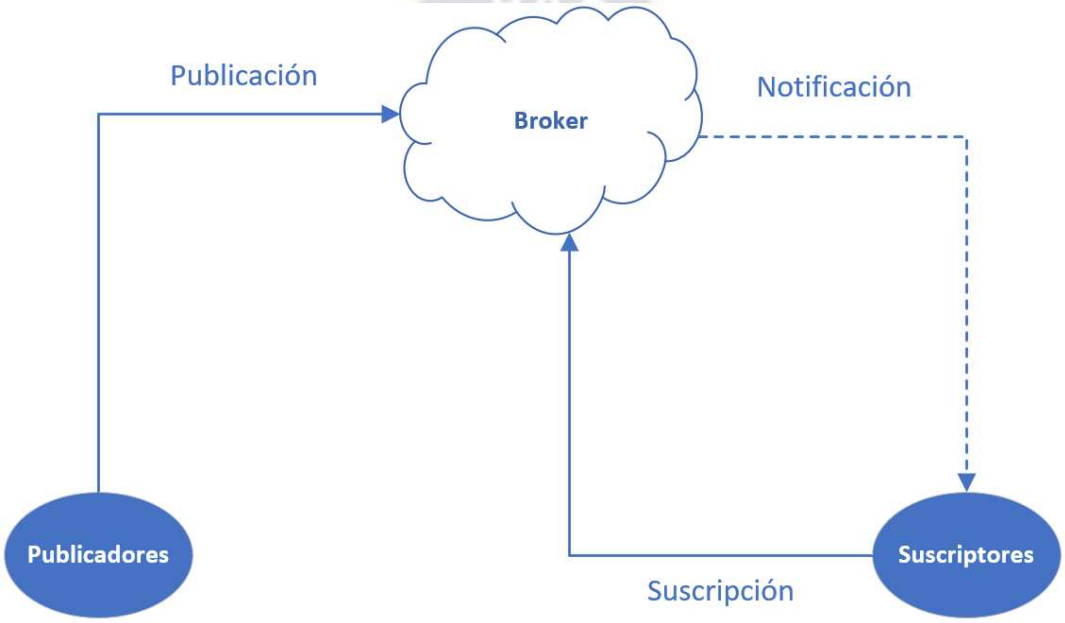
En esta sección estudiaremos los patrones utilizados en aplicaciones IoT, que son: comunicación solicitud-respuesta y publicación-suscripción.

2.2.1. Publicación/suscripción

Este patrón permite al interesado suscribirse a eventos futuros que pueden ocurrir en un sistema. Las consultas se almacenan de forma anticipada o proactiva para que puedan ser entregadas a tiempo a los interesados una vez ocurrido un evento de interés. Este patrón está compuesto de tres partes (García Davis, 2019), tal como se muestra en la figura 1:

- Los publicadores son entidades capaces de generar información y que se publica en el servicio de notificaciones.
- Los suscriptores son entidades interesadas por los eventos o patrones de eventos producidos por los editores. Expresan su interés en eventos particulares a través de una suscripción y son notificadas más tarde por el servicio de notificación cuando un evento coincide con su interés registrado.
- El servicio de notificación (*broker*) se encarga de procesar las publicaciones de eventos, registrar suscripciones y distribuir información de eventos a los suscriptores (notificación).

Figura 1. Patrón de comunicación publicación/suscripción



Fuente: Adoptado de García Davis (2019).

Las principales ventajas de este patrón son (Publish–subscribe pattern, 2022):

- Los editores están vagamente vinculados a los suscriptores y ni siquiera necesitan saber de su existencia. Incluso los editores y suscriptores pueden permanecer ignorantes de la topología del sistema.
- En muchos sistemas publicación/suscripción no es necesario que los editores y suscriptores estén conectados al servicio de notificaciones al mismo tiempo.
- La escalabilidad de este patrón puede ser mejor que la del patrón solicitud-respuesta en aplicaciones que no sean del tipo empresarial de alto desempeño.

Y como desventajas de este patrón se tiene (Publish–subscribe pattern, 2022):

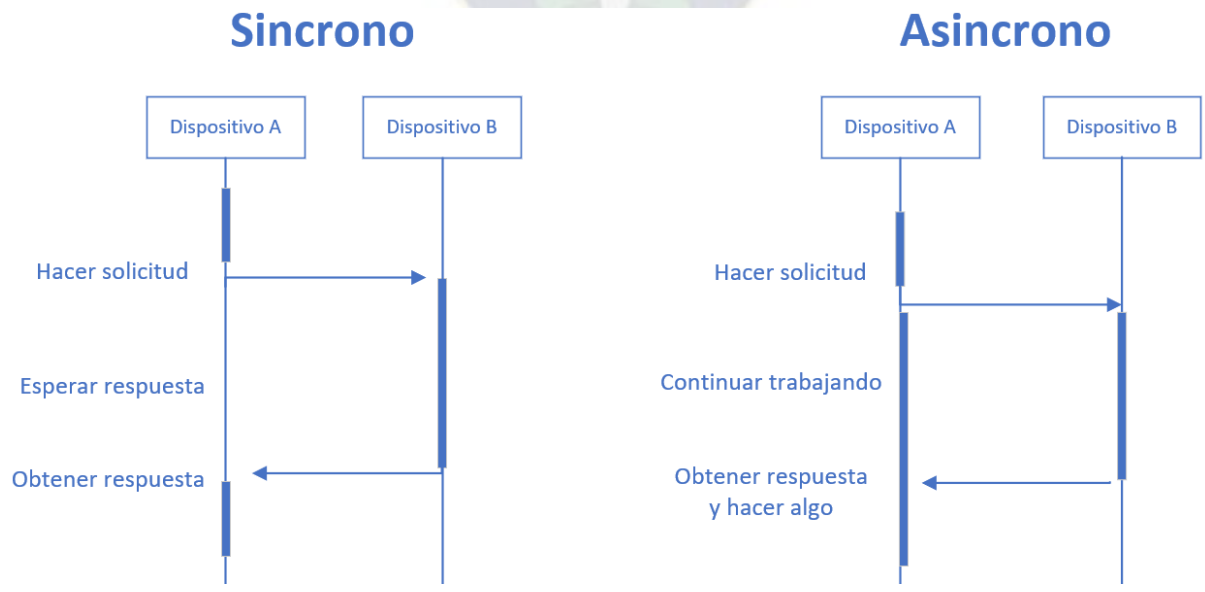
- El servicio de notificación puede estar diseñado para entregar mensajes durante un tiempo específico, pero luego dejar de intentar la entrega, ya sea que haya recibido o no la confirmación de la recepción exitosa del mensaje por parte de todos los suscriptores.
- En algunas situaciones un editor puede suponer que un suscriptor esta activo, cuando en realidad no es así, dejando abierta la posibilidad de que se pierda información, debido a que los editores desconocen el estado del suscriptor.

2.2.2. Solicitud/respuesta

El patrón de comunicación solicitud/respuesta consiste en que un dispositivo envía una solicitud de algunos datos a otro dispositivo y este responde a la solicitud. Se enfoca en el intercambio de mensajes, en el que un solicitante envía un mensaje de solicitud a un sistema, que recibe y procesa la solicitud, y finalmente devuelve un mensaje como respuesta. Permite que dos aplicaciones tengan una conversación bidireccional entre si a través de un canal, es muy común en arquitecturas del tipo cliente servidor (Request–response, 2022).

Este patrón puede ser implementado en forma sincrónica, como en el caso de llamadas de servicio web por HTTP el cual mantiene una conexión abierta y espera hasta que se entrega la respuesta o expira el periodo de tiempo de espera. Como también puede ser implementado de forma asincrónica devolviendo una respuesta en un momento posterior desconocido (Request–response, 2022). En la figura 2 se muestra los patrones de comunicación síncrono y asíncrono.

Figura 2. Patrón de comunicación solicitud/respuesta



Fuente: Adoptado de Hari (2018).

2.3. Modelo de referencia OSI aplicado a sistemas IoT

La comunicación en sistemas IoT puede describirse a través del modelo de interconexión de sistemas abiertos OSI. Este modelo proporciona un estándar para comunicar diferentes sistemas informáticos y divide el sistema de comunicación en siete capas apiladas verticalmente, cada capa cumple diferentes funciones en el intercambio de información (Modelo OSI, 2022). En la figura 3 se muestra este modelo OSI.

Figura 3. Modelo OSI

Fuente: Tomado de Barbosa (2015).

Muchas veces las capas del modelo OSI pueden nombrarse directamente por su posición en la pila, así la capa de aplicación puede ser nombrada como capa siete.

La función de cada capa puede resumirse en (Modelo OSI, 2022):

a) Capa de aplicación: es la capa donde las aplicaciones pueden acceder a los servicios las otras capas. También define los protocolos que utilizan las aplicaciones para intercambiar datos.
b) Capa de presentación: asegura que los datos se encuentren en un formato adecuado y también se encarga de cifrar los datos y comprimirlos.
c) Capa de sesión: mantiene y controla las conexiones entre dos dispositivos y es responsable del control de puertos y sesiones. Algunas veces los servicios de esta capa pueden prescindirse parcial o totalmente.
d) Capa de transporte: transporta los datos libres de errores del dispositivo de origen al de destino usando protocolos de transmisión.
e) Capa de red: define el enrutamiento existente entre una o varias redes.

f) Capa de enlace: se encarga del direccionamiento físico entre dos dispositivos que se encuentran en una misma red. Esta capa también se encarga del acceso al medio, detección de errores y del control del flujo.
g) Capa física: define la topología de red y las conexiones globales de un dispositivo hacia la red. Esta capa es la más baja y transmite la información a través del medio físico bit a bit.

En cada capa del modelo OSI se encuentran trabajando protocolos los cuales cumplen las funciones de cada capa. En los siguientes apartados se describen los protocolos más utilizados en aplicaciones del tipo IoT.

2.4. Estado del arte de los protocolos para la capa de aplicación utilizados en la IoT

2.4.1. Advanced Message Queuing Protocol (AMQP)

AMQP es un protocolo para mensajerías, sus principales características son: la orientación del mensaje, la puesta en cola (*queuing*), el enrutamiento, exactitud y la seguridad (Advanced Message Queuing Protocol, 2021).

Es un protocolo de transferencia de mensajes de propósito general que puede ser aplicable y conveniente para muchos tipos de infraestructura *middleware* de mensajería y también para el envío de mensajes punto a punto. Se caracteriza por ser bidireccional que permite a cada parte en una conexión actual iniciar transferencia, y tiene una extensibilidad y características de anotación prácticamente en todas las capas (Akilli, 2018).

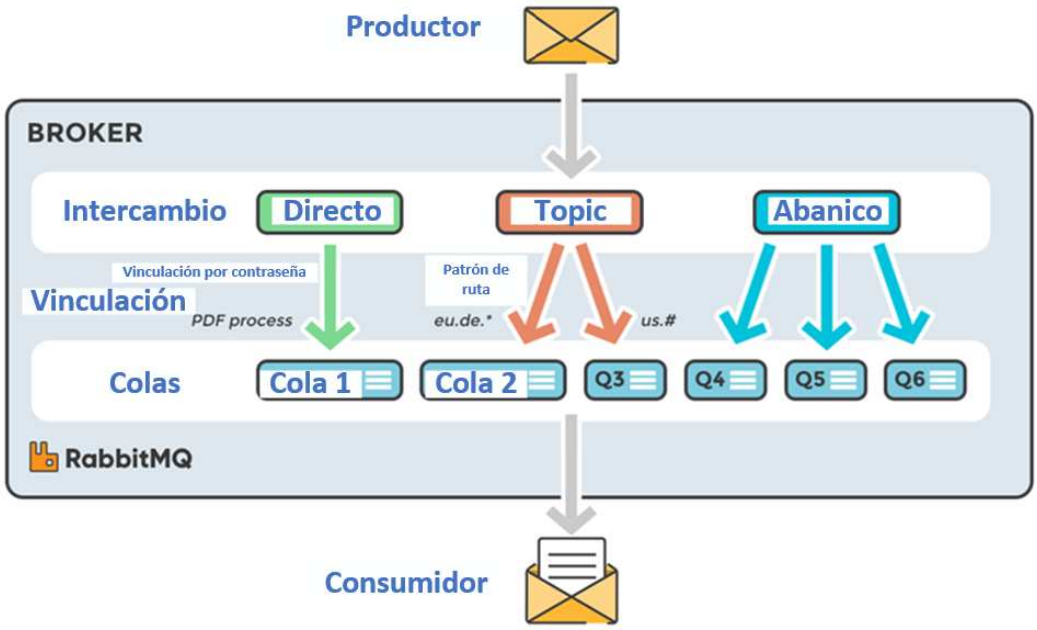
Este protocolo es principalmente utilizado para interacciones de tipo servidor a servidor o puerta de enlace a servidor, en vez de las interacciones de tipo dispositivo a dispositivo o dispositivo a puerta de enlace, esto es porque los servidores o los dispositivos que hacen de puerta de enlace son computacionalmente suficientes, y a que procesar AMQP requiere más potencia de procesamiento (Akilli, 2018).

La principal aplicación que se da a AMQP son del tipo mensajería para negocios. En la IoT el protocolo puede ser usado para el control de plano o funciones de análisis basadas en servidor (Akilli, 2018).

2.4.1.1. Arquitectura AMQP

La arquitectura de los sistemas AMQP está compuesto por un *broker* de mensajería que cuenta con un intercambio (*exchange)*, donde se conectan los productores de mensajes, y las colas que se vinculan a los *exchanges* a través de diferentes criterios. Los consumidores de los datos se conectan a las colas para recibir los mensajes que producen los publicadores (Bassi, 2021). En la figura 4 se muestra la arquitectura AMQP.

Figura 4. Arquitectura AMQP



Fuente: Tomado de Johansson (2019).

Las entidades que participan en la arquitectura AMQP son: el *broker* que es un servidor de mensajería donde se conectan los clientes y se encarga de distribuir los mensajes, los usuarios se conectan al *broker* (*consumers* y *producers*), la conexión física sobre un protocolo de transporte está relacionada a un usuario, y la conexión lógica está ligada a la conexión física (Bassi, 2021).

El *broker* se compone por intercambiadores, colas y vinculaciones. Los intercambiadores reciben los mensajes y los envían a las diferentes colas. Existen cuatro tipos de intercambiadores (Johansson, 2019).

- En el intercambio directo los mensajes son enviados a las colas cuya clave de enlace sea igual a la clave de enrutamiento del mensaje.
- En el intercambio por topic un mensaje se envía a las colas comparando la clave de enrutamiento del mensaje con el patrón de enrutamiento (vinculación).
- En el intercambio por abanico se enrutan los mensajes a todas las colas vinculadas a él.
- En el intercambio por cabecera se usan los atributos del encabezado del mensaje para el enrutamiento.

Las colas cuentan con un nombre el cual puede ser definido por un cliente o por el *broker* de forma automática. Las colas son fragmentos de memoria de tipo volátil o no volátil. Las colas no volátiles persisten después de que el *broker* se reinicia, aunque no se

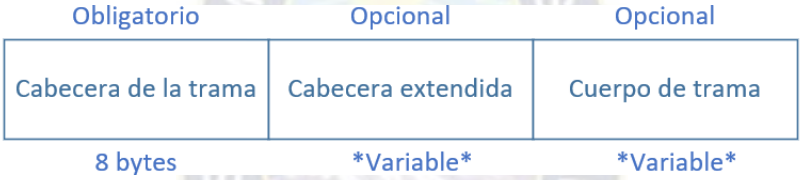
garantizan que los mensajes se guarden permanentemente, dependiendo de la configuración de los mensajes (Bassi, 2021).

La relación entre una cola y un intercambiador se denomina vinculación, este se encarga de especificar como fluye el mensaje. Vinculación mantiene las aplicaciones que producen aisladas de las aplicaciones que consumen, posteriormente se pueden realizar las configuraciones dentro del *broker* para enrutar los mensajes adecuadamente sin cambiar las aplicaciones productoras o consumidoras (Bassi, 2021).

2.4.1.2. Formato de mensajes

Las tramas se componen en tres distintas áreas: un encabezado de longitud fija, un encabezado de longitud variable y un cuerpo de longitud variable (OASIS, s.f.-c). En la figura 5 se muestra el formato de trama en sistemas AMQP.

Figura 5. Formato de trama en sistemas AMQP



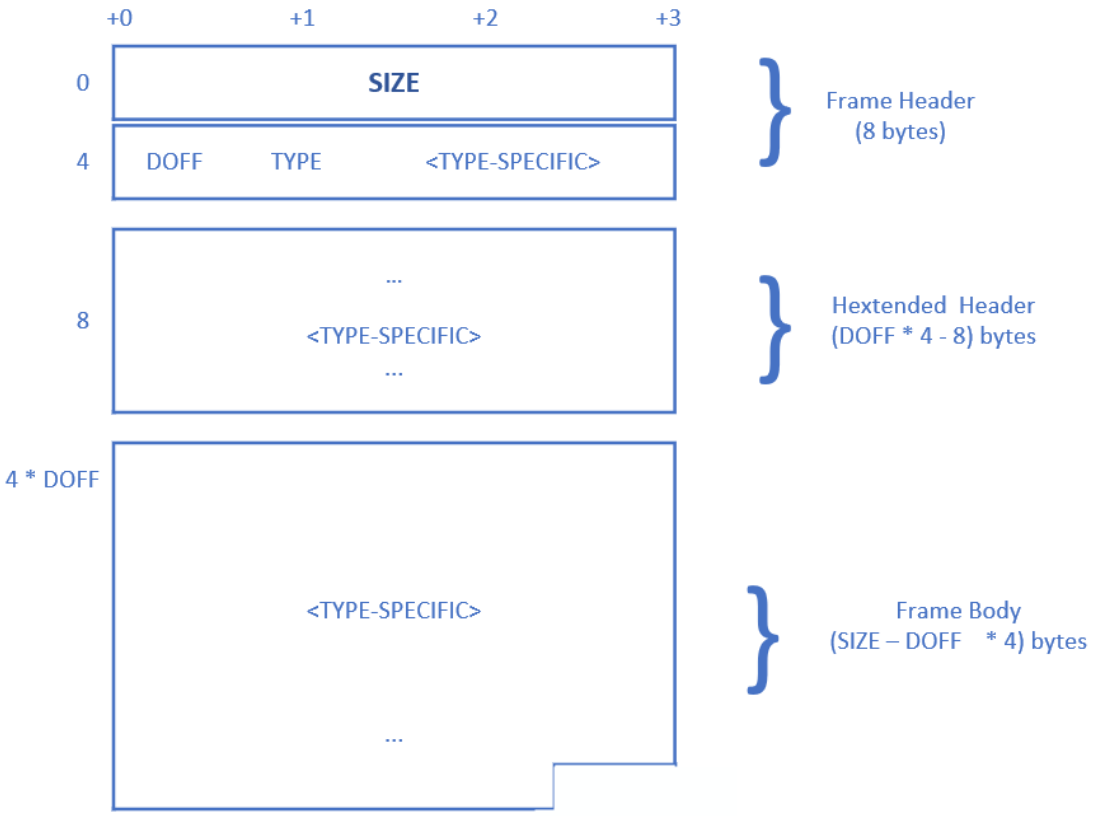
Fuente: Adaptado de (OASIS, s.f.-c).

Los 3 componentes de una trama son (OASIS, s.f.-c):

- La cabecera de la trama tiene una longitud fija de 8 bytes el cual siempre precede a cada trama, incluye información obligatoria necesaria para analizar el resto de la trama como el tamaño y el tipo.
- La cabecera extendida tiene una longitud variable el cual precede al cuerpo de la trama. Es un punto de expansión definido para futuras expansiones y depende del tipo de trama.
- El cuerpo de la trama es de longitud variable y depende de la longitud de trama.

La figura 6 se muestra el diseño general de cualquier tipo de trama (OASIS, s.f.-c):

Figura 6. Diseño de una trama general en sistemas AMQP



Fuente: Tomado de OASIS (s.f.-c).

La cabecera de la trama se compone de (OASIS, s.f.-c):

a) Longitud (*size*): ocupa los bytes 0 a 3 de la cabecera y contienen el tamaño total de la trama, es un numero entero de 32 bits sin signo que contiene el tamaño de la cabecera, la cabecera extendida y el cuerpo de la trama.
b) Desplazamiento de datos (*data offset*): ocupa el byte 4 de la cabecera e indica la posición del cuerpo dentro de la trama, su valor es un entero de 8 bits sin signo que especifica un recuento de palabras de 4 bytes.
c) Tipo (*type*): el byte 5 del encabezado es un código de tipo, que indica el formato y propósito de la trama.

Los subsiguientes bytes en el encabezado de la trama pueden interpretarse de manera diferente dependiendo del tipo de trama. Un código de tipo 0x00 indica que la trama es de tipo AMQP, un código de tipo 0x01 indica que la trama es de tipo SASL (OASIS, s.f.-c).
En las tramas de AMQP los bytes 6 y 7 contienen el número de canal. El cuerpo de la trama AMQP se define como un performativo seguido de una carga útil opaca, este

performativo esta codificado como un tipo descrito en el sistema de tipos AMQP. Los bytes restantes en el cuerpo de la trama forman la carga útil. Se debe mencionar que la presencia y el formato de la carga útil están definidos por la semántica del performativo dado (OASIS, s.f.-c).

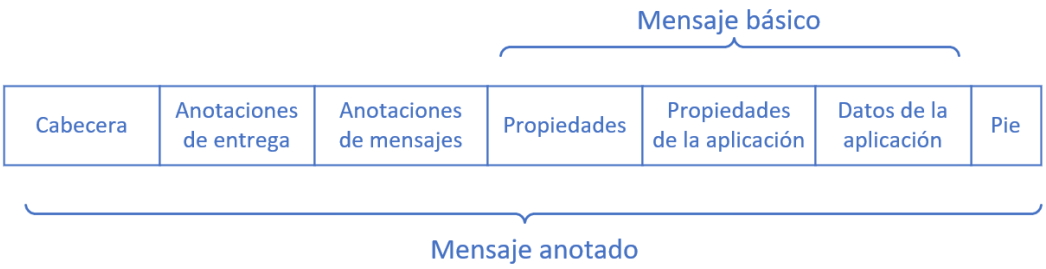
Un resumen de las funciones de cada performativo, puede ser el siguiente (AMQP: Introducing the Advanced Message Queuing Protocol, 2019):

- Abrir (*Open*) negocia los parámetros de conexión entre el intermediario (*broker*) y el cliente.
- Empezar (*Begin*) indica que se está iniciando una conexión.
- Adjuntar (*Attach*), el mensaje se adjunta con un enlace que es necesario para utilizar la transferencia de datos.
- Flujo (*Flow*) cambia el estado de un enlace.
- Transferencia (*Transfer*) el mensaje real se transmite con la trama de transferencia.
- Disposición (*Disposition*) un marco de disposición proporciona información sobre los cambios en la entrega de información.
- Despegar (*Detach*) elimina el enlace.
- Fin (*End*), indica que la conexión será terminada.
- Cerrar (*Close*), finaliza la conexión y declara que no se enviaran más tramas.

Los mensajes son transportados como carga útil en tramas que contienen el performativo de transferencia. Los mensajes pueden fragmentarse en varias tramas de tipo transferencia a fin de enviar estos mensajes en forma completa (OASIS, s.f.-c).

Se define como mensaje básico a los mensajes proporcionados por el remitente y como mensaje anotado a los mensajes vistos por el receptor (véase figura 7). El mensaje básico está conformado en tres partes: propiedades estándar, propiedades de la aplicación y datos de la aplicación (cuerpo). Un mensaje anotado incluye secciones para anotaciones al principio y al final del mensaje simple. Las anotaciones pueden viajar con el mensaje indefinidamente o ser “consumidas” por el siguiente nodo (OASIS, s.f.-c).

Figura 7. Formato de mensajes AMQP



Fuente: Adaptado de OASIS (s.f.-c).

2.4.1.3. Flujo de mensajes

AMQP está estructurado como una serie de métodos agrupados en clases, estas clases solo son agrupaciones lógicas de métodos AMQP. La clase *exchange* incluye las siguientes operaciones, métodos: exchange.declare, exchange.declare-ok, exchange.delete, exchange.delete-ok, entre otros (véase figura 8). Los pares de operaciones exchange.declare y exchange.declares-ok, así como también exchange.delete y exchange.delete-ok, son pares de operaciones de tipo petición enviada por el cliente y respuesta enviada por el *broker* (AMQP 0-9-1 Model Explained, s.f.-d).

Figura 8. Metodos exchange.declare y exchange.declare-ok en AMQP

Fuente: Tomado de AMQP 0-9-1 Model Explained (s.f.-d).

La operación exchange.declare lleva varios parámetros que permiten especificar el nombre de intercambio, tipo, el indicador de durabilidad, entre otros. Si la operación es exitosa el broker responderá con el método exchange.declare-ok. La operación exchange.declare-ok solo lleva el ID del canal (AMQP 0-9-1 Model Explained, s.f.-d).

En AMQP no es recomendable mantener muchas conexiones TCP abiertas al mismo tiempo porque consumen muchos recursos y las configuraciones del *firewall* se complican. Es por eso que las conexiones se multiplexan en canales, estos canales hacen de conexiones ligeras que comparten una única conexión TCP. Cuando se emplean estos canales cada método de protocolo también lleva una ID de canal (AMQP 0-9-1 Model Explained, s.f.-d).

Para el par de métodos queue.declare y queue.declare-ok la secuencia es similar al par de métodos exchange.declare y exchange.declare-ok (AMQP 0-9-1 Model Explained, s.f.-d).

Es importante aclarar que no todos los métodos tienen su contraparte. Algunos métodos, por ejemplo: basic.publish, no tienen su correspondiente respuesta y otros métodos como basic.get tienen más de una posible respuesta (AMQP 0-9-1 Model Explained, s.f.-d).

La mayoría de las conexiones de tipo AMQP son de larga duración y trabajan sobre el protocolo TCP. Estas conexiones utilizan autenticación y pueden ser protegidas usando TLS. Si una aplicación no necesita más estar conectada al servidor, solo se debe cerrar la conexión AMQP, en vez de cerrar abruptamente la conexión TCP subyacente (AMQP 0-9-1 Model Explained, s.f.-d).

2.4.2. Hypertext Transfer Protocol 2 (HTTP/2)

HTTP es el protocolo que permite la recuperación de recursos conectados a la red, se basa en la arquitectura REST. Su primera versión fue lanzada en 1991 y hasta la fecha ha sufrido muchos cambios (Kinsta, 2020).

Es importante mencionar que Representational State Transfer (REST) se basa en la arquitectura cliente/servidor sin estado, donde cada petición debe contener toda la información necesaria a ser procesada, debido a que no se guardan estados previos. En REST la interacción entre clientes y servidores se da por un conjunto único de métodos (Londoño Delgado, 2020).

2.4.2.1. Inicio de conexión y trama HTTP/2

El protocolo HTTP/2 esta sobre una conexión TCP, usa los esquemas de HTTP/1.1 en cuanto al uso de HTTP como tal y HTTPS, redireccionando a los puertos 80 para HTTP y 443 para HTTPS. HTTP/2 cuenta con dos identificadores, uno es h2 y es utilizado cuando HTTP/2 usa TLS, y el otro es h2c que se usa cuando HTTP/2 no usa TLS (Marín Pérez, 2017).

Inicio de conexión

La obtención de recursos vía HTTP/2 desde un servidor puede realizarse de tres formas:

a) Sabiendo que el servidor si soporta HTTP/2 de antemano: la conexión puede inicializarse a nivel HTTP/2 directamente, esto puede ser mediante una lista preconfigurada en el cliente, métodos de descubrimiento alternativos, o un registro de tipo SRV anunciado por DNS. No es usado este método debido a que HTTP/2 no es el único estándar que se usa en la red (Fernández & Gabriel , 2016).
b) Desconociendo si el servidor soporta HTTP/2: en este caso el cliente usa la cabecera Upgrade con el contenido h2c (véase figura 9). También se debe incluir una línea de cabecera HTTP/2-*Settings* (Marín Pérez, 2017).

Figura 9. Petición Upgrade en HTTP

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

Fuente: Tomado de Marín Pérez (2017).

La respuesta del servidor si no soporta HTTP/2 sería (véase figura 10):

Figura 10. Respuesta Upgrade del servidor si no soporta HTTP/2

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
```

Fuente: Tomado de Marín Pérez (2017).

Pero si el servidor si soportase HTTP/2, la respuesta seria (véase figura 11):

Figura 11. Respuesta Upgrade del servidor si soporta HTTP/2

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c

[ HTTP/2 connection ...
```

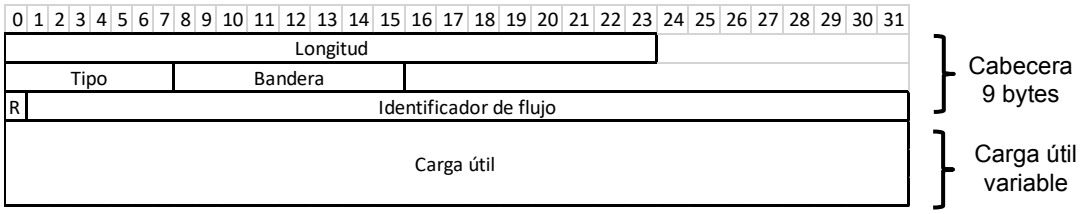
Fuente: Tomado de Marín Pérez (2017).

c) Cuando el cliente realiza una petición HTTPS desconociendo si el servidor soporta HTTP/2: el cliente usa TLS con la extensión de negociación en la capa de aplicación (ALPN). Como en el caso anterior el cliente utiliza la cabecera Upgrade en la cual se debe incluir la identificación h2 que indica el uso de TLS (Marín Pérez, 2017).

Trama HTTP/2

El formato de trama de HTTP/2 es el siguiente (véase figura 12):

Figura 12. Formato de trama HTTP/2



Fuente: Tomado de Fernández & Gabriel (2016).

Una descripción breve de los componentes de una trama puede ser la siguiente (Fernández & Gabriel , 2016):

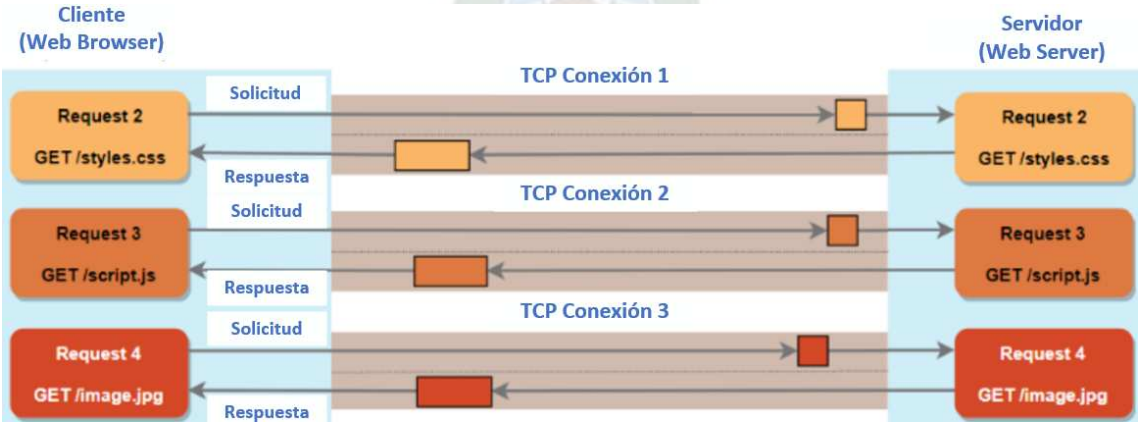
- Campo Longitud está representado por un entero de 24 bits sin signo, este campo indica la longitud del *Payload* sin contar los 9 bytes de la cabecera.
- Campo Tipo de 8 bits indica el tipo de trama.
- Campo Bandera de 8 bits indica las funciones booleanas específicas de acuerdo al tipo de trama.
- R es un bit reservado cuyo valor es 0 al ser enviado y es ignorado cuando se recibe.
- Campo Identificador de flujo está representado por 31 bits sin signo y determina a que flujo pertenece el flujo.
- La carga útil es de longitud variable y contiene la información que se desea transmitir.

2.4.2.2. Flujo de tramas

En HTTP/2 se utilizan flujos para asociar tramas bidireccionales e independientes entre el cliente y el servidor. Los flujos permiten que las tramas de distintos flujos en la misma conexión se mezclen y pueden ser creados y cerrados por el cliente o el servidor en cualquier momento. El control de flujo se realiza a través de la trama WINDOW_UPDATE, que permite al receptor establecer el ritmo de envío de bytes y actualizar la cantidad de bytes que puede recibir. También se implementan las prioridades de flujo a través de la trama PRIORITY, permitiendo al cliente construir un árbol de pesos y asignar valores a los objetos restantes en función de la prioridad (Marín Pérez, 2017).

A continuación, en la figura 13 se muestra el funcionamiento de las conexiones sin multiplexación.

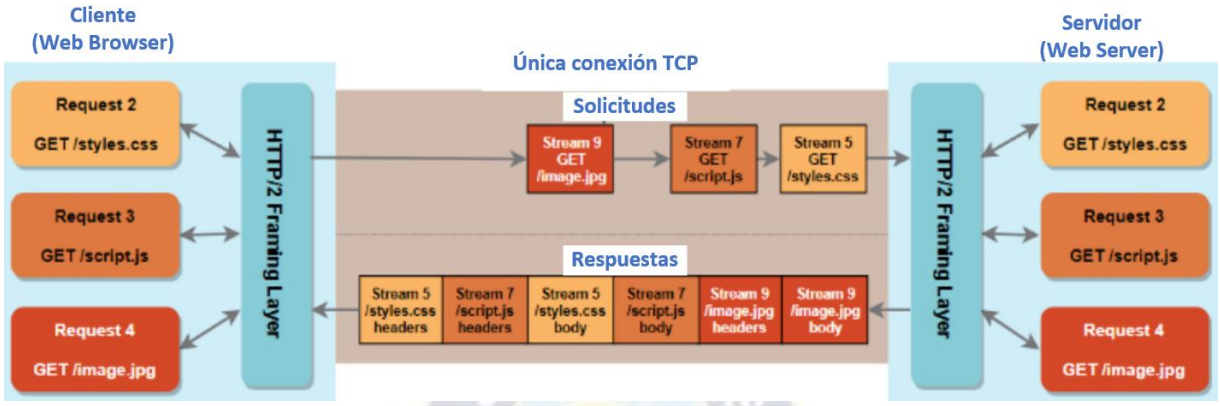
Figura 13. Conexión HTTP/2 sin multiplexación



Fuente: Adaptado de Pollard (2019).

Por otra parte, el funcionamiento de las conexiones con multiplexación es el siguiente (véase figura 14):

Figura 14. Conexión HTTP/2 con multiplexación



Fuente: Adaptado de Pollard (2019).

En la conexión sin multiplexación se debe esperar el tiempo de respuesta para obtener cada objeto requerido, este método es ordenado pero ineficiente. Mientras que, en la conexión con multiplexación, se puede ver que se solicitaron 3 objetos, se tuvo que realizar un árbol para obtener los objetos más prioritarios antes y estos se intercalan haciendo uso de la multiplexación (Marín Pérez, 2017).

Compresión de encabezado

En HTTP/2 las cabeceras experimentan compresiones, para que de esa forma se mejores los tiempos de respuesta. El algoritmo que se usa para esto es HPACK, que elimina campos de cabecera redundantes y eliminar posibles vulnerabilidades (Marreros Guzman, 2021).

HPACK usa dos técnicas para comprimir peticiones, una técnica de compresión es la de Huffman para codificar con menos bytes las cabeceras más comunes, y la otra técnica mantiene una lista de los encabezados utilizados en las anteriores solicitudes de cliente-servidor (Marín Pérez, 2017).

A continuación, en las tablas 3 y 4 se muestra cómo funciona la compresión de cabeceras:

Tabla 3. Compresión de cabecera HPACK, estado inicial

Cabecera	Dato	Índice tabla	Estado de transmisión de cabecera
:method	GET	2	SI
:host	ejemplo.com	62	SI
:path	/recurso	63	SI
user-agent	Mozilla/5.0	64	SI

Fuente: Adaptado de Marín Pérez (2017).

En la tabla 4, específicamente, se puede ver que solo se transmitió la nueva cabecera ":path" y se le asigno el valor 65, mientras que las otras cabeceras son transmitidas mediante los índices de la tabla (Marín Pérez, 2017).

Tabla 4. Compresión de cabecera HPACK, estado final

Cabecera	Dato	Índice tabla	Estado de transmisión de cabecera
:method	GET	2	NO
:host	ejemplo.com	62	NO
:path	/recurso_2	65	SI
user-agent	Mozilla/5.0	64	NO

Fuente: Adaptado de Marín Pérez (2017).

HTTP/2 Server Push

Server Push permite al servidor enviar información en cache adicional al cliente en forma proactiva, anticipándose a futuras solicitudes (Kinsta, 2020).

En la figura 15 se puede ver que el cliente solicita el recurso HTML y el servidor elige empujar el recurso HTML, y también los recursos CSS y IMG, en vez de esperar una petición expresa por los recursos CSS y IMG. El cliente coloca los recursos CSS y IMG en su cache para uso futuro, y de esa forma se ahorran dos solicitudes y respuestas, reduciendo de esa forma la latencia de la red (Kinsta, 2020).

Figura 15. Funcionamiento de Server Push en HTTP/2

Fuente: Adaptado de Kinsta (2020).

Protocolos binarios

HTTP/2 utiliza comandos binarios para completar ciclos de solicitud-respuesta, esto permite resolver contribuciones con el encuadre y simplifica la implementación de comandos (véase figura 16). Además, la semántica permanece constante (Kinsta, 2020).

Figura 16. Protocolos binarios en HTTP/2

Fuente: Adaptado de Kinsta (2020).

En la imagen se describe el funcionamiento de esta implementación, el navegador haciendo uso del convertidor binario de HTTP/2 se encarga de convertir los comandos de texto a binario antes de ser transmitidos por la red (Kinsta, 2020).

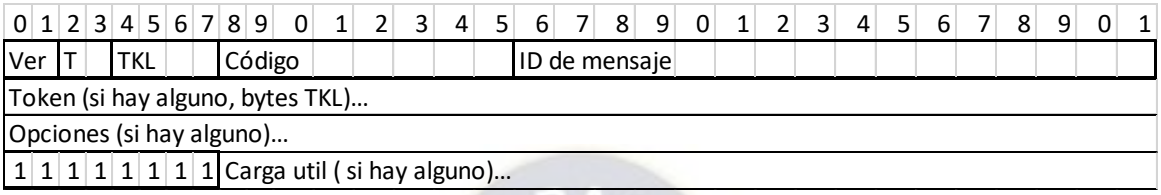
2.4.3. Constrained Application Protocol (CoAP)

CoAP es un protocolo de web orientado hacia *machine-to-machine*, diseñado para nodos y redes de bajo consumo de energía, procesamiento y memoria limitados. Se basa en la arquitectura REST y permite el descubrimiento de recursos mediante URI y Content-Type. Utiliza conexión UDP con fiabilidad opcional, intercambio asíncrono de mensajes, bajo *overhead* y sencillez en el mapeo de HTTP. Los mensajes se envían de forma asíncrona sobre UDP en el modelo cliente/servidor (Gimeno Gimenez, 2013).

2.5.3.1. Formato de mensaje

Los mensajes están formados por una cabecera de longitud fija, una cantidad variable de opciones y una carga útil (*payload),* como se muestra en la figura 17, los dos últimos campos que se mencionaron no son obligatorios (Gimeno Gimenez, 2013).

Figura 17. Formato de mensaje CoAP

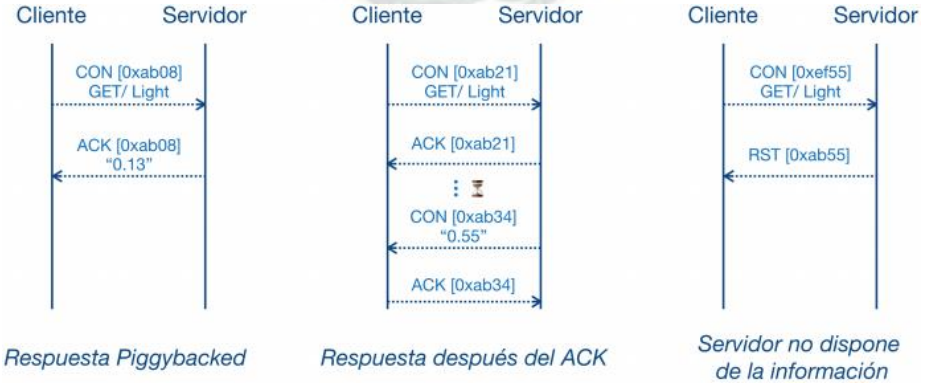


Fuente: Tomado de Shelby et al. (2018).

En el protocolo CoAP existen 4 tipos de encabezados, los cuales son (Bassi, Introducción al protocolo CoAP, 2021):

a) Mensaje *Confirmable* (CON): se lo utiliza cuando se necesita garantizar que el mensaje llegue al destinatario. Una vez que se transmita este mensaje se espera recibir un *acknowledge* (ACK) con el mismo ID de la petición, a esta situación se denomina *piggybacked*. La respuesta se envía luego en otro mensaje del tipo CON y otro ID distinto, el servidor responde con un ACK vacío a la petición, hasta que se tenga la información solicitada. Finalmente, si el servidor no cuenta con la información responderá con un mensaje de tipo RST (véase figura 18).

Figura 18. Mensaje CON en COAP

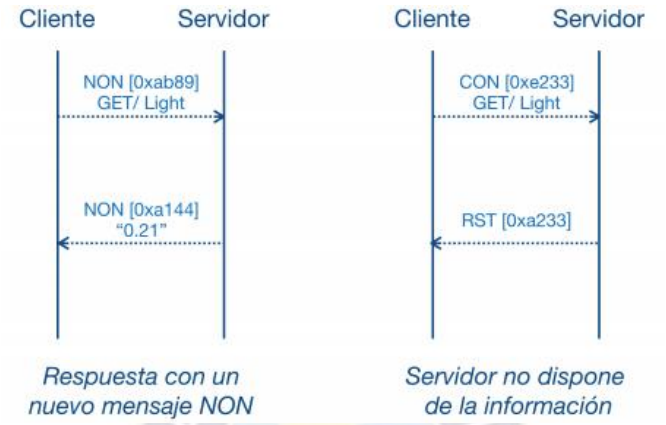


Fuente: Tomado de Bassi (2021).

b) Mensaje *Non-Confirmable* (NON): no requiere confirmación por parte del receptor, y la información solicitada será enviada en otro mensaje del tipo NON. Este tipo de mensajes no son tan fiables, sin embargo, tienen

identificación única. Si el servidor no puede responder con la información solicita enviara el mensaje RST (véase figura 19).

Figura 19. Mensaje NON en COAP



Fuente: Tomado de Bassi (2021).

c) Mensaje Acknowledgement (ACK): mensaje que reconoce que llego un mensaje conformable especifico identificado por su ID de transacción. Puede tener *payload*, como en el caso de *piggybacked*, y algunas opciones para proporcionar más detalles al respecto.
d) Mensaje de reinicio (RST): este tipo de mensaje indica que se recibió un mensaje especifico, pero no puede ser procesado correctamente debido a la falta de información de contexto.

El ID de transacción es número entero única el cual es mantenido por un nodo para los mensajes CON y NON que envía. Este ID se utiliza para hacer coincidir un mensaje de tipo ACK con una solicitud pendiente, para transmitir y también descartar mensajes duplicados (Bassi, Introducción al protocolo CoAP, 2021).

Cabecera de mensaje

La cabecera de mensaje es indispensable en los mensajes ya que define el tipo de mensaje, si es *request* con un determinado método o si es un mensaje response con un código que indica si la petición ha sido tratada correctamente y entendida por el servidor o si hubo un problema. Los campos que forman la cabecera son (Gimeno Gimenez, 2013):

- Ver indica la versión de CoAP.
- T indica el tipo de mensaje que puede ser: CON, NON, ACK o RST.
- TKL (*Token Length*) indica la longitud del campo Token.
- Código indica si el mensaje se trata de una petición (valores de 1-31), de una respuesta (64-191) o si está vacío (0). Si es una petición, los métodos pueden

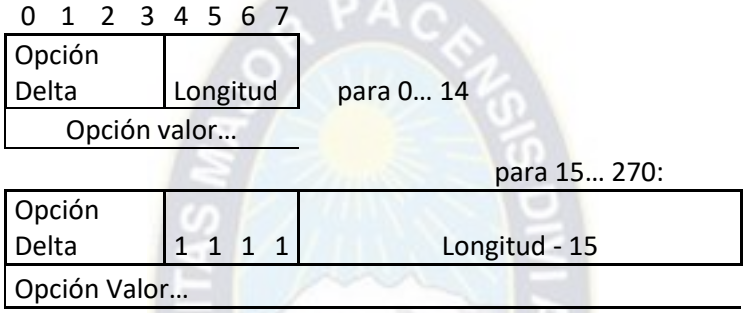
ser: GET, POST, PUT o DELETE y si es una respuesta el método es *Response Code*.

- ID de mensaje sirve para identificar al emisor del mensaje.

Opciones de mensaje

Las opciones de un mensaje vienen después del token, se ordenan *mediante Option Numbers*, los *Option Numbers* impares que corresponden a opciones críticas y los pares a opcionales (véase figura 20). Se diferencian las opciones crítica y opcional en cómo se trata cuando se recibe una opción no reconocida (Gimeno Gimenez, 2013).

Figura 20. Opciones de mensajes CoAP



Fuente: Tomado de Shelby et al. (2018).

Cada campo se describirá a continuación (Gimeno Gimenez, 2013):

- Opción Delta: valor numérico que indica la diferencia entre Opción Número de la opción actual y la anterior.
- Longitud, indica el tamaño en bytes del campo Opción Valor, generalmente ocupa 4 bits, pudiendo indicar un valor de 0 a 14 bytes. Si este campo vale 15, se añade otro byte para así codificar tamaños del Opción Valor de 15 a 270 bytes.
- Opción Valor: su tamaño y formato depende de la opción propiamente contenida. Contiene el valor concreto que toma la opción.

Las opciones definidas en el protocolo CoAP se puede ver en el Anexo 1.

Carga útil

La carga útil de cada mensaje contiene métodos de tipo solicitud (*request)* que un cliente puede solicitar, estos métodos pueden ser (Maceda Dal-re, 2016):

a) GET: sirve para obtener la información del recurso identificado por el URI del *request*. En caso de éxito el servidor devuelve un código 2.05 (*content*) o 2.03 (*valid*) como respuesta. Este método es de carácter idempotente, por lo que si se repite el resultado es el mismo.

b) POST: sirve para crear un nuevo recurso en el servidor. Si el recurso se crea correctamente, el código de respuesta deberá ser 2.01 (*created*) o 2.04 (*changed*), incluyendo el URI del nuevo recurso. Este método no idempotente
c) PUT: actualiza el recurso identificado por el URI del *request*, si ese URI tiene un recurso, el cuerpo del mensaje debe tener las modificaciones del recurso y debería retornar una respuesta 2.04 (*changed*) o 2.01(*created*). Este método es idempotente
d) DELETE: solicita que se elimine el recurso identificado por el URI del *request*. En caso que se pueda eliminar correctamente el recurso, el servidor debería devolver una respuesta 2.02 (*deleted*). Esta función es idempotente.

Cabe aclarar que URI (*Uniform Resource Identifier*) es una cadena de caracteres que identifica los recursos de una red de forma univoca (Identificador de recursos uniforme, 2022).

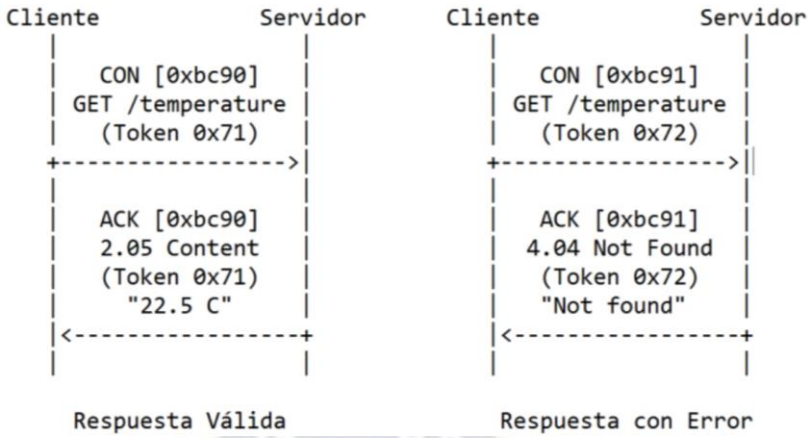
2.5.3.2. Flujo de mensajes

CoAP utiliza métodos de solicitud GET, PUT, POST y DELETE para el intercambio de mensajes asíncronos entre dos nodos. El nodo cliente envía solicitudes a uno o más recursos alojados en el nodo servidor, y este responde indicando el éxito o no de la petición recibida. Las respuestas dependen del tipo de mensaje en el que se envió el método de solicitud y pueden ser de las clases de código de respuesta 2.xx (*Success*), 4.xx (*Client Error*) y 5.xx (*Server Error*). CoAP también utiliza mensajes CON y NON para el envío de solicitudes (Hervas Parra, 2018).

Para el acuse de recibo de los mensajes por parte del servidor, puede haber dos tipos de técnicas para la respuesta:

a) *Piggy-Backing:* ocurre cuando el servidor responde inmediatamente a una solicitud de tipo CON y envía un mensaje de tipo ACK. Este mensaje ACK puede indicar éxito o fallo al tratar la solicitud (véase figura 21).

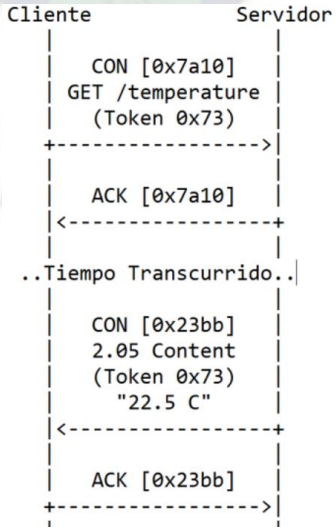
Figura 21. Respuesta tipo *Piggy-Backing* CoAP



Fuente: Tomado de Hervas Parra (2018).

b) *Separate*: se utiliza cuando al recibirse una petición en el servidor y esta no puede responder en forma inmediata debido a que no dispone en ese momento de acceso al recurso o porque se encuentra saturado. Si la petición se recibe a través de un mensaje de tipo CON y el servidor no puede responder inmediatamente, el servidor responderá con un mensaje ACK que confirma que recibió una petición y que será tratada a la brevedad posible. Mas adelante, la respuesta con el contenido del recurso se envía en un mensaje de tipo CON, en el cual el cliente deberá confirmar que ha recibido correctamente la respuesta (véase figura 22).

Figura 22. Respuesta tipo *Separate* en CoAP



Fuente: Tomado de Hervas Parra (2018).

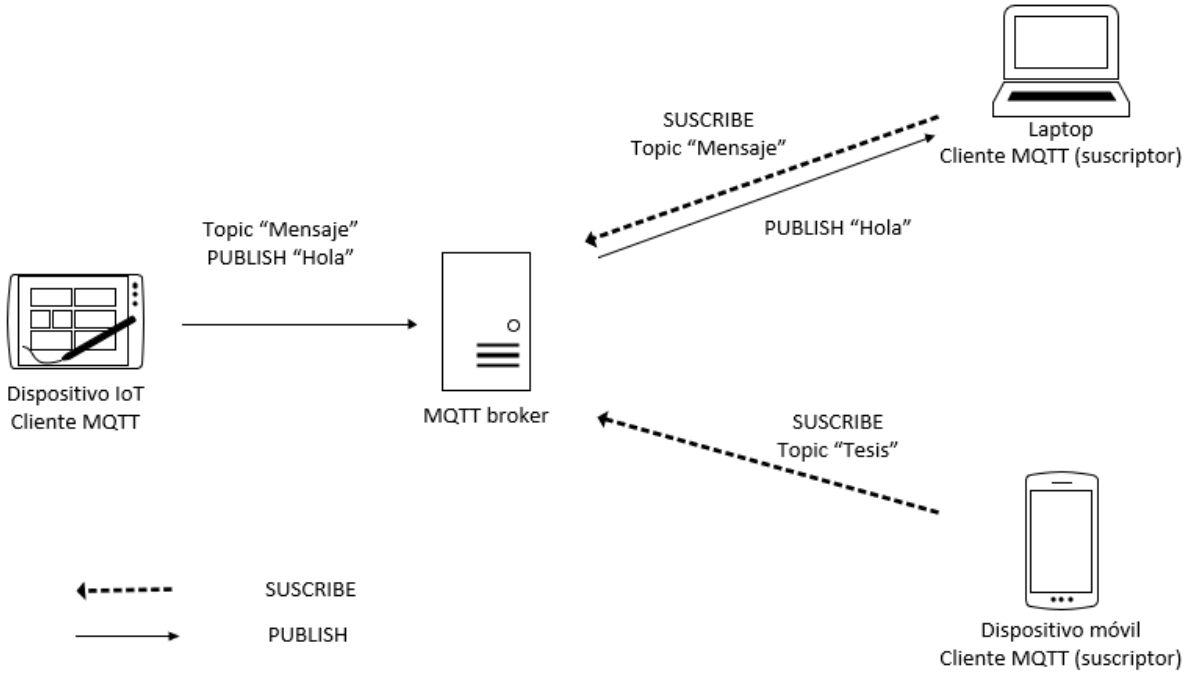
2.5.4. MQ Telemetry Transport (MQTT)

MQ Telemetry Transport, conocido anteriormente como Message Queing Telemetry Transport, es un protocolo inventado por el doctor Andy Standford-Clark de IBM y Arlen Nipper de Arcom (Eurotech) en 1999 y ahora es un estándar abierto OASIS (MQTT, 2022).

Es uno de los protocolos más usados en la IoT. Fue diseñado específicamente para sistemas restringidos por su mínimo ancho de banda y los pocos recursos que consume, sin descuidar aspectos tales como la confiabilidad y cierto grado de garantía de entrega. Se basa en el patrón de comunicación publicación/suscripción y trabaja en la capa de aplicación que funciona en la parte superior de la pila TCP/IP (Salas, 2018).

En MQTT se tienen clientes, que cuentan con un código ID único denominado clientID, los cuales se conectan a un *broker*. En la figura 23 se puede ver que un dispositivo IoT envía un mensaje “Hola” al *topic* “Mensaje”, el *broker* es el responsable de enviar el mensaje a los dispositivos que están suscritos al *topic* “Mensaje” (Salas, 2018). Cabe hacer notar que no existe una comunicación directa entre clientes y que cada mensaje tiene un *topic*.

Figura 23. Arquitectura MQTT



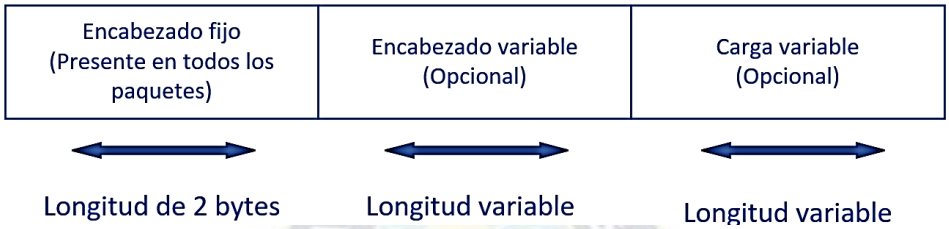
Fuente: Adaptado de Salas (2018, pág. 7).

La función del *broker* es la de recibir los mensajes de los publicadores, luego filtrarlos según el *topic*, para finalmente enviarlos a sus subscriptores correspondientes (Portas, 2021).

2.5.4.1. Formato de paquete

Los paquetes MQTT están compuestos por tres partes: encabezado fijo, encabezado variable y carga útil. El encabezado fijo tiene una longitud de dos bytes y está presente en todos los paquetes, mientras que el encabezado variable y la carga útil son optativos y su longitud es variable (véase figura 24). Por la naturaleza del encabezado variable y la carga útil, los paquetes MQTT tiene una longitud variable (BytesofGigabytes, 2019).

Figura 24. Formato de paquete MQTT



Fuente: Adaptado de BytesofGigabytes (2019).

La estructura del encabezado fijo es la siguiente (véase figura 25):

Figura 25. Encabezado fijo de paquete MQTT

7	6	5	4	3	2	1	0	bit
Message type				Header Flags				1er Byte
Remaining Length								2do Byte...

Fuente: Adaptado de BytesofGigabytes (2019).

El *message type* representa el tipo de solicitud de conexión, este se define mediante un valor y tiene una longitud de cuatro bits (BytesofGigabytes, 2019). En el Anexo 2 se puede observar los tipos de mensajes MQTT.

Existen tres tipos de indicadores *header flags*, los cuales son: DUP, QoS y RETAIN (véase figura 26).

Figura 26. Estructura de los indicadores de encabezado en MQTT

3	2	1	0	bit
DUP	QoS	QoS	RETAIN	Header Flags

Fuente: Adaptado de BytesofGigabytes (2019).

El indicador DUP (duplicado) ocupa el bit tres, si su valor es 0 quiere decir que este es el primer intento del cliente de enviar el paquete de control de publicación, y si su valor es 1 significa que el cliente está intentando de nuevo enviar el paquete enviado anteriormente (BytesofGigabytes, 2019).

El indicador QoS de calidad de servicio indica la garantía de entrega de mensajes PUBLISH, ocupa dos bits y existen tres diferentes opciones que son QoS 0, 1 y 2.

El indicador RETAIN (retención), si su valor es igual a 1 significa que el *broker* retendrá o almacenará el paquete a menos que no haya ningún suscriptor con el mismo *topic* que el *topic* en el paquete almacenado. Tan pronto como haya un suscriptor, el *broker* entregara el paquete almacenado. Si RETAIN es igual a 0 el *broker* no retendrá el paquete (BytesofGigabytes, 2019).

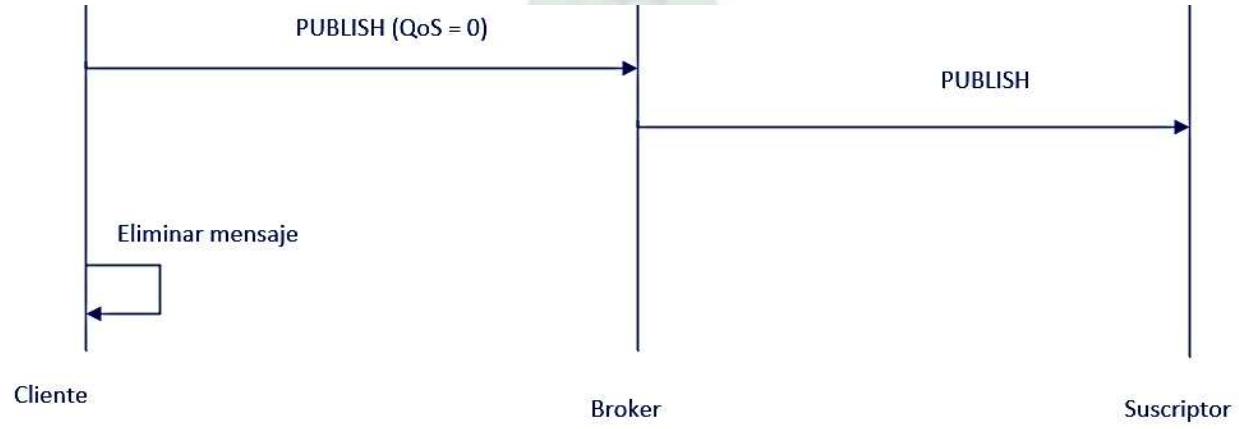
La *remaining length* (longitud restante), es un campo utilizado para identificar la longitud del paquete MQTT, puede tener una longitud de uno a cuatro bytes (BytesofGigabytes, 2019). El valor que puede tomar es de 0 a 127 y está definido por los siete bits inferiores, el bit más significativo (bit identificador) si es igual a 0 quiere decir que no hay un byte posterior y si es igual a 1 quiere decir que hay otro byte para identificar la longitud del paquete. La longitud máxima que puede describir este campo es de 256 MB y solo incluye la longitud del encabezado variable y la carga variable (Solace, 2022).

2.5.4.2. Flujo de mensajes

El protocolo MQTT permite diferentes niveles de envió de mensajes, los cuales dependen del indicador QoS.

Cuando el indicador QoS es igual a 0 (como máximo una vez) quiere decir que el cliente que publica envía un mensaje al *broker* y lo elimina después (véase figura 27). No se enviará ningún mensaje de *acknowledgement* (reconocimiento) al cliente que publica, por lo que este cliente que publica no sabrá si el mensaje llego al *broker*. Este indicador QoS ofrece la misma garantía de entrega que el protocolo TCP (Salas, 2018).

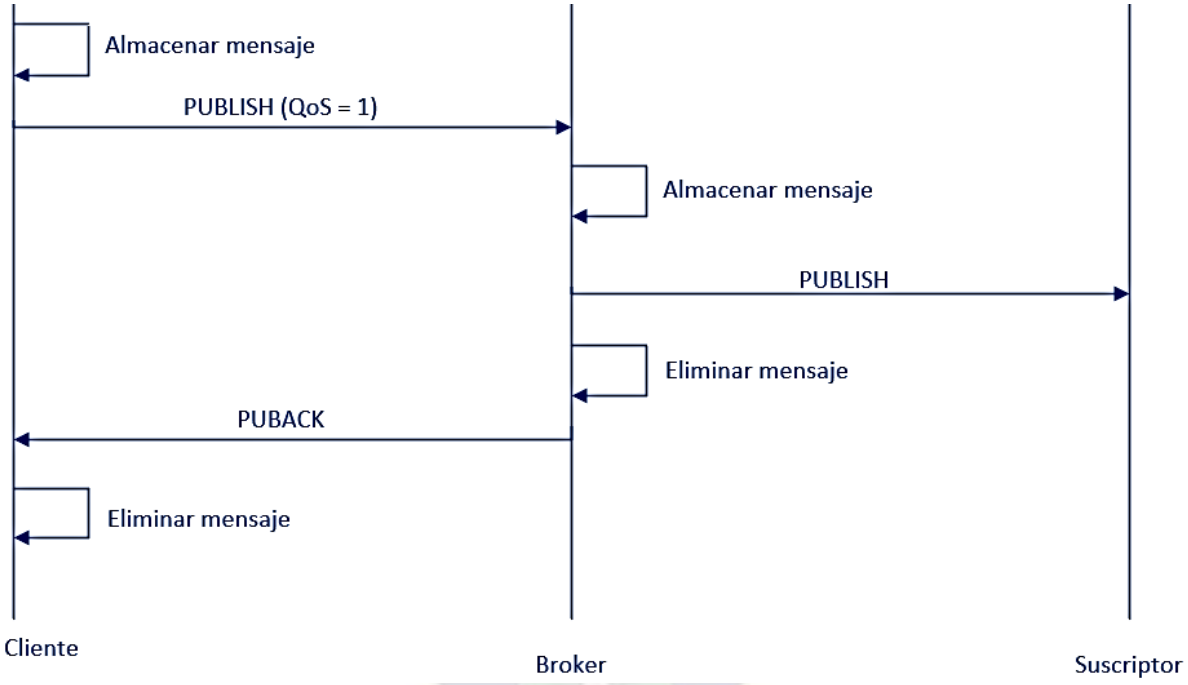
Figura 27. QoS 0 en MQTT



Fuente: Adaptado de Salas (2018, pág. 10).

Cuando el indicador QoS es igual a 1 (al menos una vez) en este caso cuando el cliente envía o publica un mensaje a un *topic*, este mensaje será almacenado por el cliente y se mantendrá esperando por un *acknowledgement* del *broker*. Si eso no ocurre el cliente intentará publicar el mensaje de nuevo. Solo cuando se reciba el *acknonowledgement* del *broker* (PUBACK), el cliente eliminará el mensaje (Salas, 2018). Para este indicador es posible que exista mensajes duplicados enviados (véase figura 28).

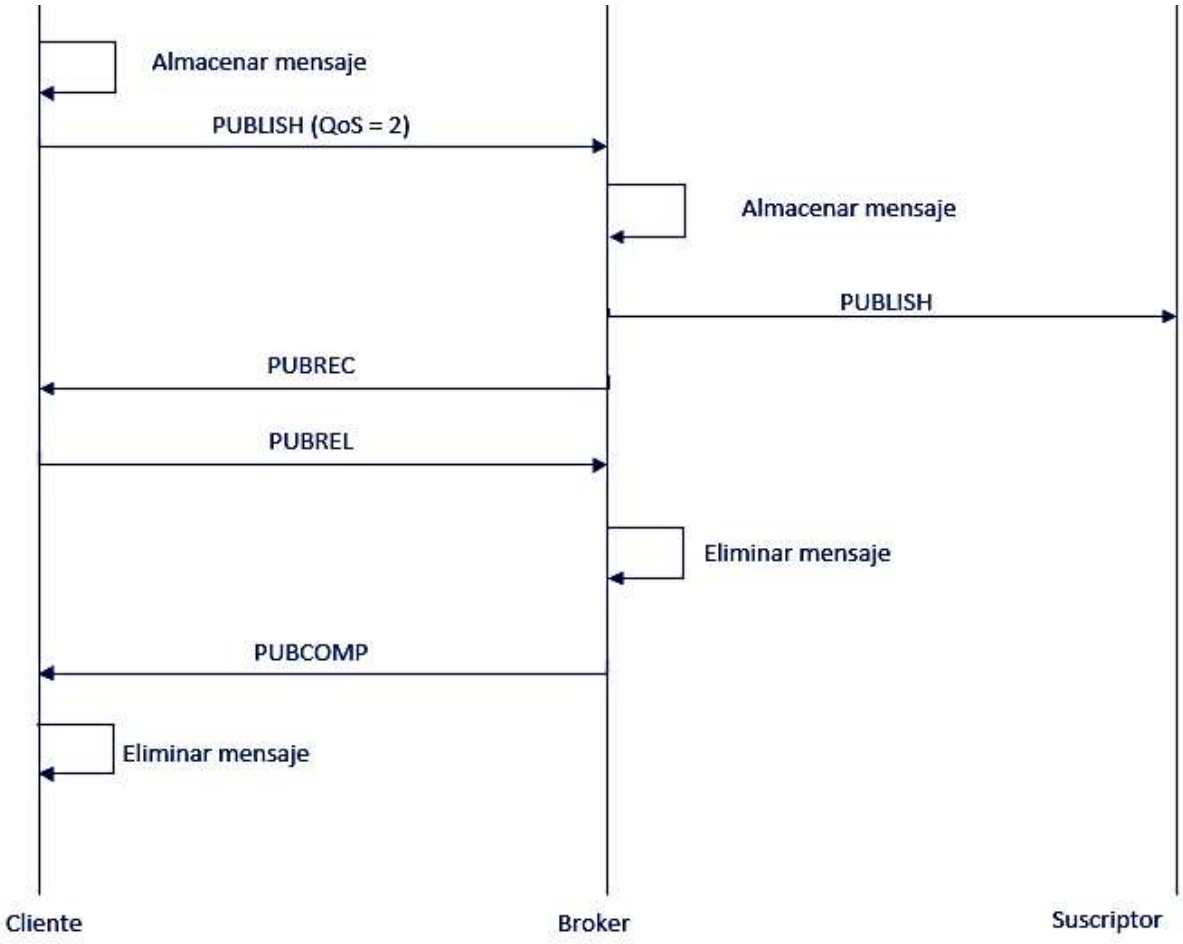
Figura 28. QoS 1 en MQTT



Fuente: Adaptado de Salas (2018, pág. 11).

Si el indicador QoS es 2 (exactamente una vez) el cliente almacenará el mensaje para luego publicarlo al *broker*. El *broker* también almacenará el mensaje hasta que un suscriptor a este mensaje se conecte. Después de haber publicado el mensaje el *broker* enviará un mensaje al cliente un mensaje de publicación recibida (PUBREC). Luego el cliente responderá con un comunicado publicado (PUBREL) y recién el *broker* borrará el mensaje previamente almacenado y enviará una publicación de completado (PUBCOMP) al cliente para que sepa que el mensaje efectivamente llegó y se puede borrar (véase figura 29). Este indicador de calidad de MQTT es el más lento (Salas, 2018).

Figura 29. QoS 2 en MQTT



Fuente: Adaptado de Salas (2018, pág. 11).

3. METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Métodos de investigación

El método de investigación empleado es el experimental, que es un tipo de método cuantitativo, este método se utiliza para probar hipótesis y establecer relaciones causales entre variables. En el presente trabajo se ha utilizado un diseño experimental para comparar el rendimiento de protocolos para la capa de aplicación utilizados en la Internet de las Cosas. Se ha realizado pruebas en diferentes escenarios, y ha medido métricas como la latencia, la perdida de paquetes y el rendimiento (throughput).

3.2. Tipo de investigación

Los tipos de investigación empleados son el descriptivo y explicativo. Descriptivo por que se describe cómo funcionan los protocolos de la capa de aplicación en la Internet de las Cosas en diferentes escenarios. Los datos recopilados al implementarse la técnica de evaluación de desempeño se analizarán para proporcionar una visión del desempeño de los protocolos.

Mientras que el enfoque explicativo se utiliza para comprender las causas detrás de los resultados de desempeño de los protocolos. Al analizar los resultados de las pruebas experimentales se identificará las relaciones causales entre las características especificas de los protocolos y su desempeño.

3.3. Universo o población de estudio

El universo de estudio está conformado por los protocolos para la capa de aplicación utilizados en la Internet de las Cosas.

En particular se eligió los protocolos AMQP y MQTT, en sus variantes de calidad de servicio 0 (como máximo una vez), 1 (al menos una vez) y 2 (exactamente una vez). Estos dos protocolos son de los más utilizados, y trabajan en la arquitectura de publicación y suscripción, lo que es muy provechoso al momento de compararlos.

Además de que estos protocolos cuentan con mucha bibliografía disponible, lo cual facilita su estudio e implementación.

3.4. Sujetos vinculados a la población

Según nuestro proceso de operacionalización de variables, los sujetos vinculados a la población son las métricas de evaluación, las cuales se describen a continuación:

a) Latencia: Mide el tiempo que tarda un paquete de datos en viajar desde el origen hasta el destino. En el contexto de los protocolos de la capa de aplicación en la IoT, la latencia es importante para medir la velocidad de respuesta de los dispositivos IoT.

b) Pérdida de paquetes: La pérdida de paquetes mide el porcentaje de paquetes de datos que no llegan a su destino. En nuestro caso, la pérdida de paquetes puede ser un indicador de la eficiencia y fiabilidad de los protocolos.
c) Rendimiento (*throughput*): Mide la cantidad de datos que se pueden transferir en un período de tiempo determinado. Para nuestro estudio, el *throughput* es importante para medir la capacidad de los protocolos para transferir datos de manera eficiente y rápida.

3.5. Fuentes y diseño de los instrumentos de relevamiento de información

3.5.1. Fuentes de la Investigación

Las fuentes de la investigación es el tiempo que toma en llegar a los paquetes a su destino para diferentes tamaños de los paquetes haciendo uso de diferentes protocolos para la capa de aplicación utilizados en la Internet de las Cosas. Otra fuente de investigación es la cantidad de paquetes que llegaron exitosamente a su destino.

Según la revisión bibliográfica que se hizo, es importante considerar los niveles de calidad que pudieran tener los protocolos, ya que estos influyen mucho en el desempeño de los protocolos.

3.5.2. Diseño de los instrumentos de relevamiento de la información

Para el diseño de los instrumentos de relevamiento de información se revisó bibliografía relacionada con Linux y Python para la automatización del envió de paquetes y la medición de los parámetros a ser estudiados

Se emplearon herramientas, tales como la librería “time” de Python la cual permite registrar tiempos de envío de paquetes. Además, debido a la necesidad de que los tiempos de los dispositivos a utilizarse estén con los relojes sincronizados se hizo uso del protocolo NTP (*Network Time Protocol*).

Cabe mencionar que toda la información recolectada se la almacenó en archivos de tipo CSV para facilitar su procesamiento y análisis.

3.6. Procesamiento y análisis de la información

Para el procesamiento y análisis de la información se hizo uso de la estadística descriptiva, a través de la biblioteca pandas para Python. Pandas permite calcular y visualizar gráficamente estadísticos descriptivos de los datos que se quiere procesar.

Los estadísticos descriptivos que se calcularon son: media, mediana, desviación estándar, mínimos y máximos. Con esta información se tendrá un panorama más claro del comportamiento de las métricas que se consideran para determinar el desempeño de los protocolos.

4. MARCO PRÁCTICO

En esta sección se desarrollará el procedimiento para la implementación de la técnica de evaluación de protocolos para la capa de aplicación en la IoT.

Se diseñarán las pruebas experimentales que se utilizarán para evaluar los protocolos de la capa de aplicación. En este proceso se planificará el procedimiento experimental que se llevará a cabo.

Siguiendo al diseño de las pruebas experimentales, se implementará los protocolos, para luego efectuar las pruebas y almacenar las mediciones. Esta etapa toma tiempo y es en la que se debe tener cuidado en el manejo de los datos para garantizar la validez de los resultados.

Finalmente, se presentarán los hallazgos obtenidos en las pruebas realizadas. Se analizarán y compararán los resultados de las mediciones de latencia, pérdida de paquetes y rendimiento (*throughput*) para cada protocolo y sus niveles de calidad de QoS, con el fin de determinar cuál es el protocolo y en qué nivel de calidad de servicio presenta un mejor desempeño.

4.1. Metodología de desarrollo de la solución

4.1.1. Diseño de pruebas experimentales

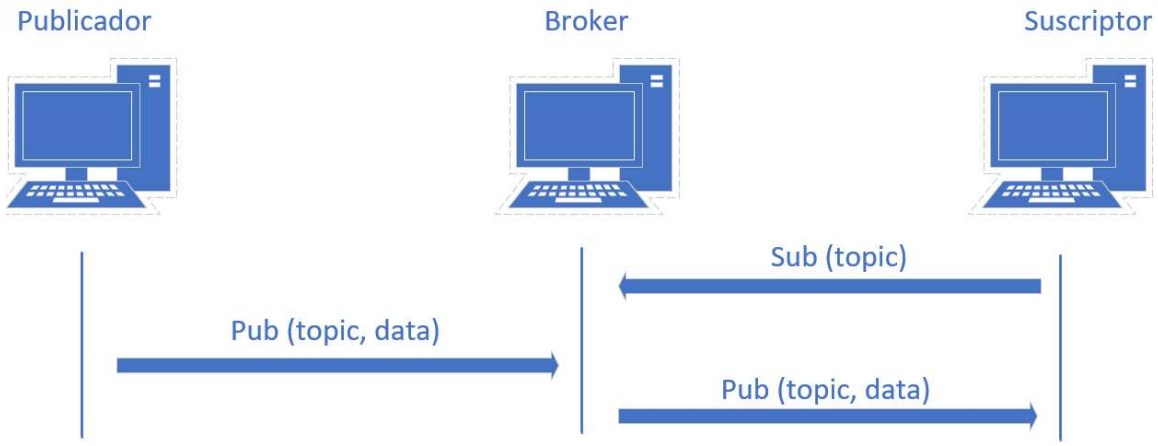
En esta sección se diseñan las pruebas experimentales para evaluar los protocolos de la capa de aplicación en la IoT.

4.1.1.1. Escenario de pruebas experimentales

Los experimentos se llevaron a cabo en una red local WiFi domiciliaria utilizando un modem Hitron CGNV5-MAX con velocidad de datos de 300 Mbps y un servicio de internet de 35 Mbps de bajada y 15 Mbps de subida proporcionado por el proveedor de servicios de internet.

Debido a que los protocolos son del tipo publicación/suscripción la red está conformada por tres máquinas virtuales con el sistema operativo Ubuntu, una máquina virtual hará de publicador, otra maquina hará de *broker* y la última hará de suscriptor (véase figura 30).

Figura 30. Escenario de pruebas experimentales



Fuente: Elaboración propia.

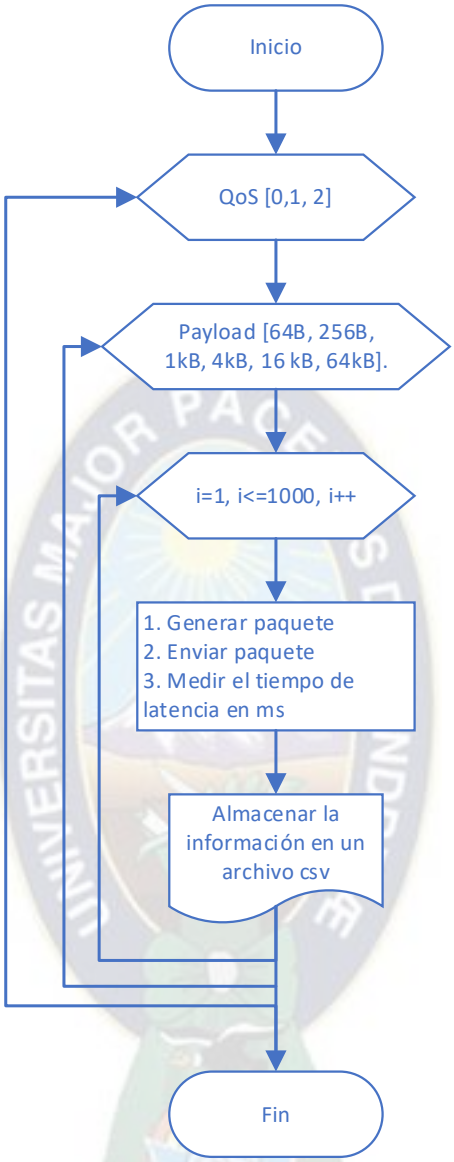
4.1.1.2. Definición de los casos de prueba

Para las tres métricas que se evaluarán se definieron las siguientes pruebas.

Latencia

Para evaluar la latencia (ms), se enviarán repetidamente paquetes de datos de diferentes tamaños de *payload* desde los 64B hasta los 64kB, en un total de 1000 repeticiones por tamaño para garantizar obtener una muestra estadísticamente significativa de los resultados. Durante las pruebas se considerará la calidad de servicio (QoS) de la red. El siguiente diagrama de flujo resume el diseño de la prueba de latencia (véase figura 31).

Figura 31. Diagrama de flujo para la prueba de latencia

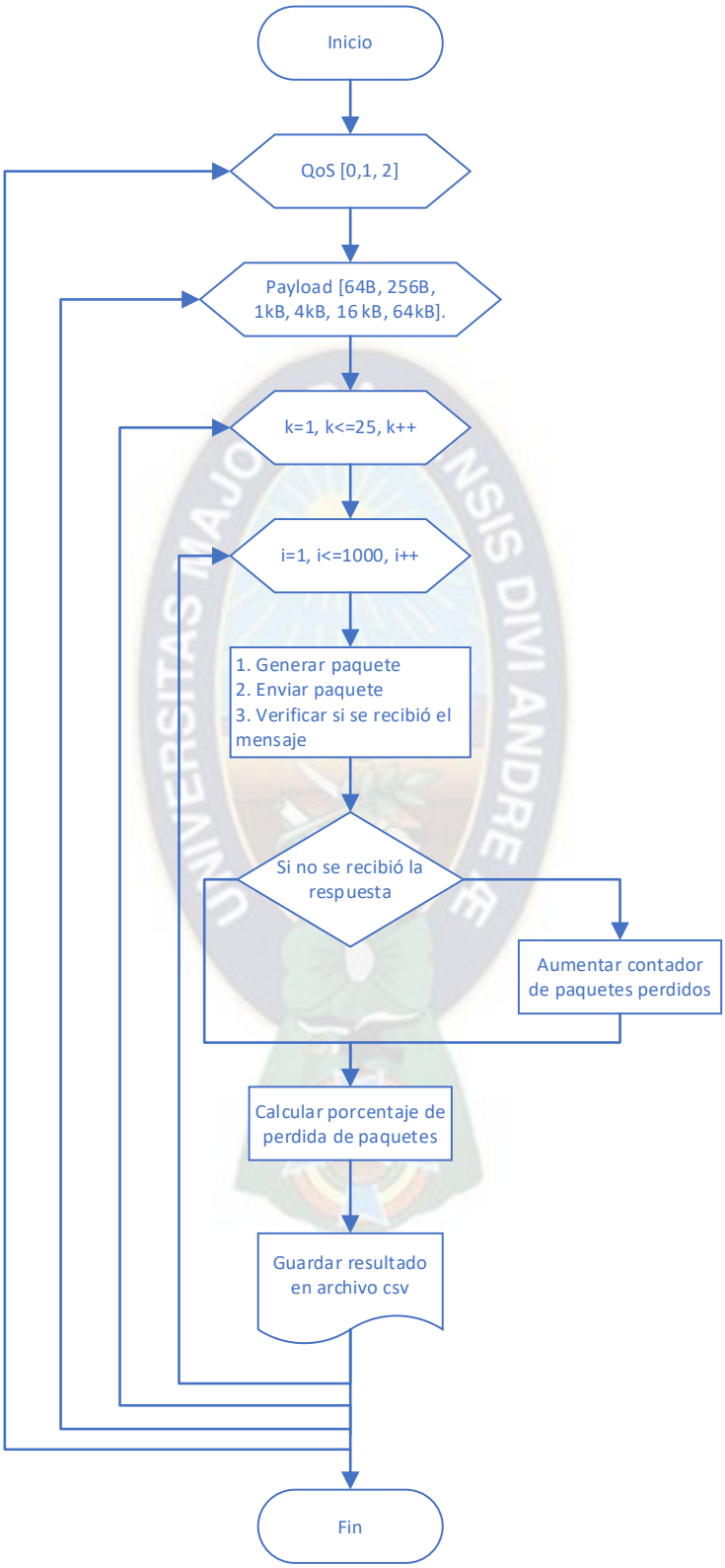


Fuente: Elaboración propia.

Perdida de paquetes

En este caso se evaluará la pérdida de paquetes, se realizará 25 veces el envío de 1000 paquetes para cada tamaño (64 B, 256B, 1kB, 4kB, 16kB y 64 kB) y calidad de servicio (QoS) disponibles. Para cada repetición, se medirá el número de paquetes perdidos y se calculará el porcentaje de pérdida. Los resultados de cada prueba se almacenarán en archivos separados. El siguiente diagrama de flujo resume el proceso (véase figura 32).

Figura 32. Diagrama de flujo para la prueba de perdida de paquetes

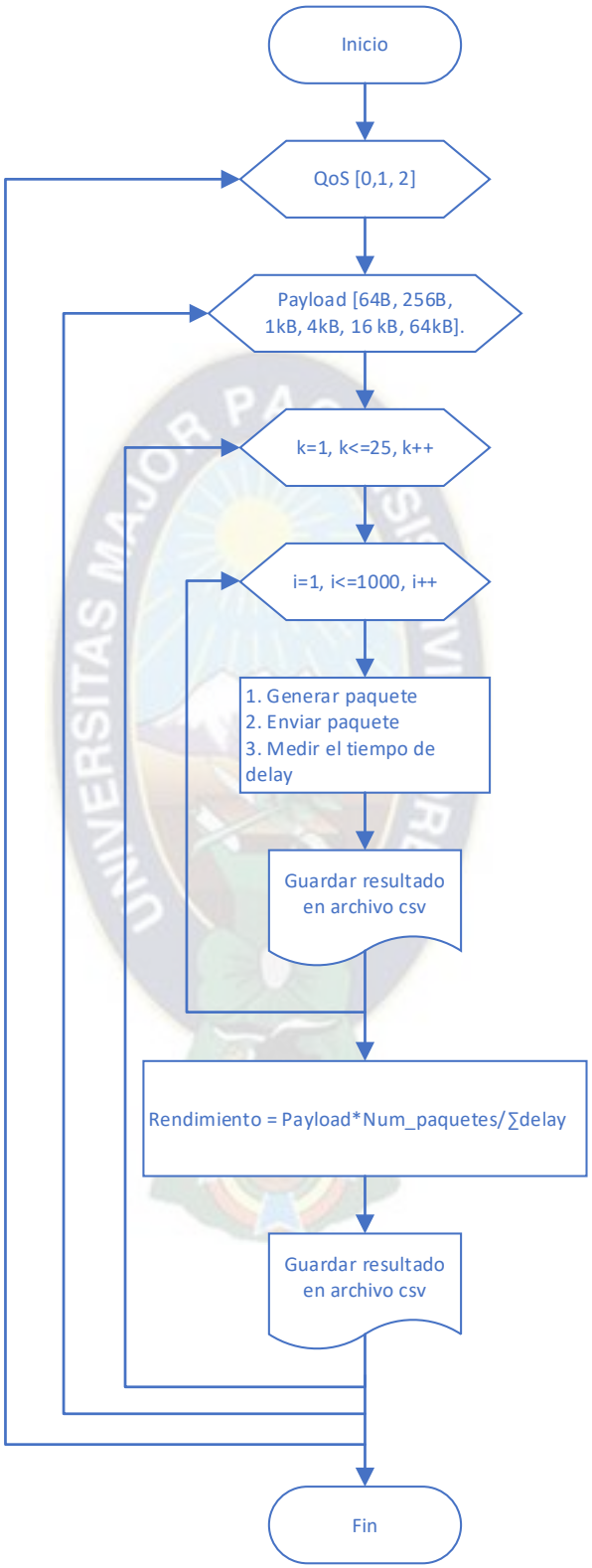


Fuente: Elaboración propia.

Rendimiento (*throughput*)

Mientras que para el estudio del rendimiento se seguirá el mismo procedimiento que se elaboró en la perdida de paquetes. La diferencia es que en este caso se medirá el rendimiento (*throughput*), para eso se medirá la cantidad de información enviada exitosamente entre el tiempo total que toma a esta información en ser enviada. El diagrama de flujo que se muestra a continuación refleja el proceso (véase figura 33).

Figura 33. Diagrama de flujo para la prueba de rendimiento



Fuente: Elaboración propia.

4.1.2. Implementación de protocolos de prueba

Antes de implementar los protocolos en las tres máquinas virtuales se debe instalar Python en estas máquinas, para eso se seguirá el siguiente procedimiento:

a) Abra el terminal de Ubuntu.

b) Actualice la lista de paquetes disponibles.

   sudo apt-get update

c) Instale el paquete de Python.

   sudo apt-get install python

Además, se hará uso del protocolo NTP (*Network Time Protocol*) que nos permite sincronizar los relojes de los dispositivos conectados a una red para que de esta forma se maneje un solo tiempo en la red y así se obtenga una sincronización de tiempo precisa. Para eso se debe seguir el siguiente procedimiento:

a) Incluir el siguiente comando en los terminales.

   sudo apt-get install ntp

b) Luego se debe iniciar el protocolo con el siguiente comando.

   sudo service ntp start

c) También podemos verificar el estado del protocolo.

   sudo service ntp status

4.1.2.1. Implementación de protocolos AMQP

Para el caso del protocolo AMQP se siguió el siguiente procedimiento en el terminal de la máquina virtual que hace de *broker*.

a) Se debe instalar la biblioteca erlang.

   sudo apt-get install erlang

b) Ahora instala el paquete de RabbitMQ.

   sudo apt-get install rabbitmq-server

c) Ahora habilitamos los servicios de RabbitMQ.

   sudo systemctl enable rabbitmq-server

d) En el servidor se deberá iniciar el servidor AMQP.

   sudo systemctl start rabbitmq-server

e) Podemos verificar si el servidor AMQP de RabbitMQ está funcionando correctamente con el comando.

   sudo systemctl status rabbitmq-server

f) Para acceder a la terminal grafica de RabbitMQ podemos ejecutar el comando.

   sudo rabbitmq-plugins enable rabbitmq_management

g) Ahora podemos ver la terminal grafica ingresando a la ip local y el puerto 15672 (véase figura 34).

Figura 34. Inicio de sesión de RabbitMQ

Fuente: Elaboración propia.

h) Creamos el usuario “admin” con contraseña “admin”.

   sudo rabbitmqctl add-user admin admin

i) Asignamos a “admin” la etiqueta de administrador.

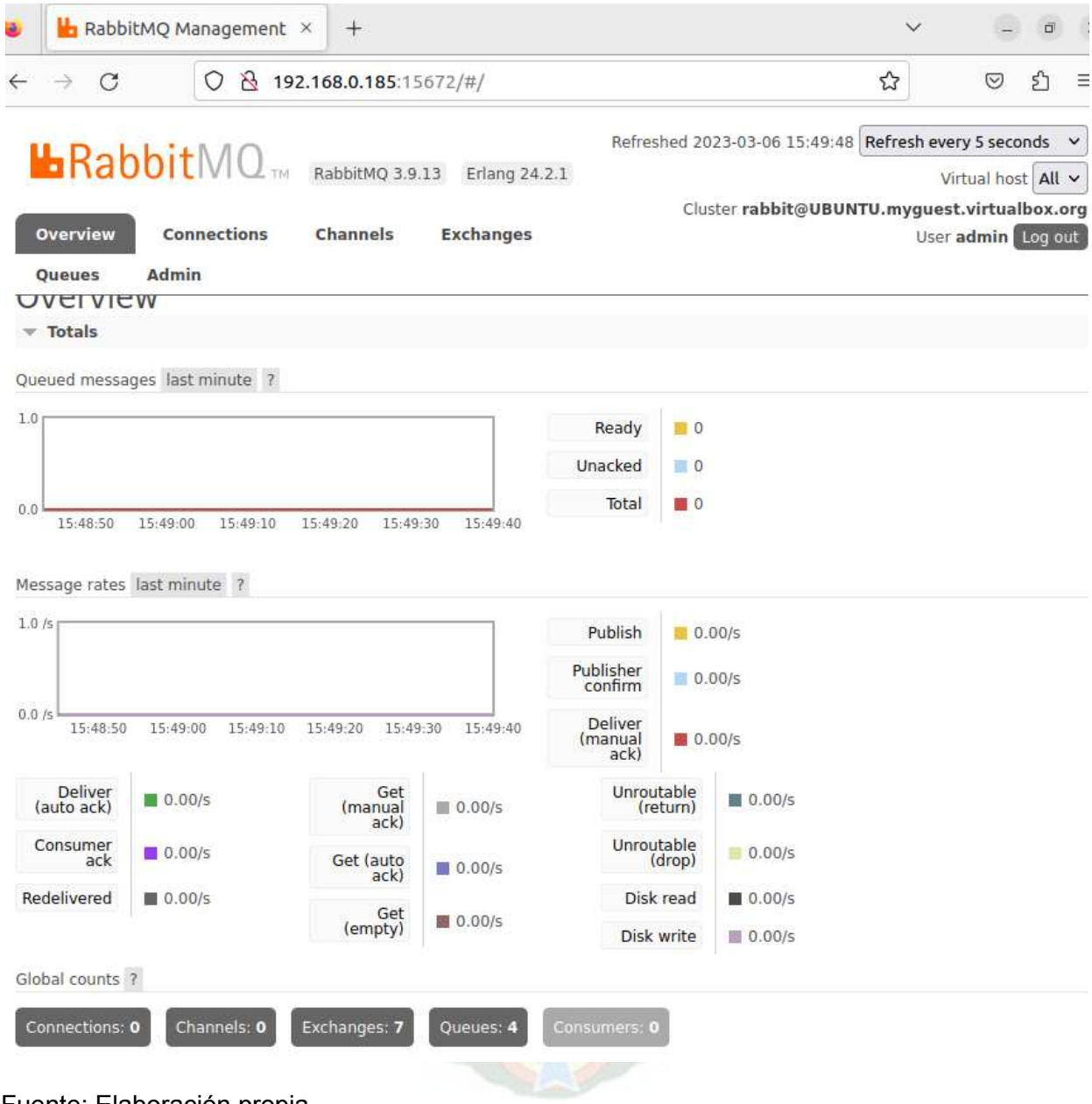
   sudo rabbitmqctl set_user_tags admin administrator

j) Ahora asignamos los permisos de lectura, escritura y ejecución.

   Sudo rabbitmqctl set_permissions -p / admin "." "." "."

k) Ingresando el usuario y contraseña en la terminal gráfica se puede ver lo siguiente (véase figura 35).

Figura 35. Terminal grafica RabbitMQ



Fuente: Elaboración propia.

Para el caso de la máquina que hace de publicador y de suscriptor se siguió los pasos a) y b) para el *broker*.

Algunas veces puede que sea necesario habilitar el puerto 15672 en el servidor o cliente, para eso se puede incluir el siguiente comando:

```
sudo ufw allow 15672/tcp
```

Para instalar Pika, y usar Python y AMQP en Ubuntu, siga los siguientes pasos tanto en el *broker* como en suscriptor y publicador:

sudo apt-get install python3-pika

4.1.2.2. Implementación de protocolos MQTT

En el caso del protocolo MQTT se sigue el siguiente procedimiento para instalar MQTT tanto para la máquina que hace de publicador, *broker* y suscriptor:

a) Instala MQTT (Mosquitto) con el siguiente comando.

   sudo apt-get install mosquito

b) Ahora escribe el siguiente comando.

   sudo apt-get install mosquito-client

   sudo apt-get install mosquito-clients

c) Podemos verificar el estado con el siguiente comando.

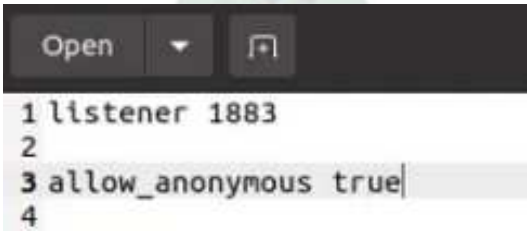
   systemctl status mosquitto.service

d) Detenemos mosquitto con la siguiente instrucción.

   systemctl stop mosquitto.service

e) Luego debemos configurar el uso del puerto 1883 para eso abrimos un editor de texto y escribimos las instrucciones (véase figura 36):

Figura 36. Configuración de puertos para el protocolo MQTT



Fuente: Elaboración propia.

f) Guardamos el documento como test.conf y en el terminal ejecutamos la siguiente instrucción

   mosquitto -c ubicación_del_archivo/test.conf -v

En algunas ocasiones puede ser que no esté habilitado el puerto 1883, así que debemos utilizar el siguiente comando:

```
sudo ufw allow 1883/tcp
```

Ahora que tenemos implementado los dos protocolos se procederá a las pruebas a la ejecución de las pruebas de desempeño.

4.1.3. Ejecución de la prueba de latencia

En esta sección se implementó el diagrama de flujo previamente desarrollado para la medición de la latencia. Para llevar a cabo las pruebas, se desarrolló código en python para automatizar las pruebas. En los Anexos 3 a 8 se presenta el código desarrollado, a continuación, se detallan las líneas de código más relevantes y su funcionalidad en el proceso de medición para cada protocolo y sus respectivos QoS.

4.1.3.1. Prueba de latencia protocolo AMQP

Los códigos tanto para la máquina que hace de publicador como la que hace de suscriptor para la prueba de latencia comienzan con:

```
import pika

import time
```

La primera línea de código importa el módulo pika que es una biblioteca de cliente de RabbitMQ para Python. Este módulo permite a crear, enviar y recibir mensajes utilizando protocolos AMQP (*Advanced Message Queuing Protocol*).

La segunda línea de código es import time importa el módulo time que proporciona funciones para trabajar con el tiempo. En particular, el módulo time incluye la función sleep que permite pausar la ejecución del programa durante un número de segundos determinado.

Luego sigue el siguiente código:

```
connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185', 5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

channel.queue_declare(queue='ejemplo')
```

Del anterior código, la primera línea de código crea una conexión de bloqueo (BlockingConnection) utilizando la dirección IP del servidor ('192.168.0.185') y el número de puerto (5672) del servidor de RabbitMQ. La conexión utiliza una instancia de

ConnectionParameters para proporcionar información sobre la conexión, incluyendo la ruta de acceso virtual ('/') y las credenciales de autenticación (PlainCredentials) que consisten en un nombre de usuario ('admin') y una contraseña ('admin').

La segunda línea de código utiliza la conexión recién creada para crear un canal (*channel*) a través del cual se pueden enviar y recibir mensajes a y desde el servidor RabbitMQ. El canal es creado con el método channel() de la conexión.

En la tercera línea de código se utiliza para declarar una cola de mensajes en RabbitMQ utilizando el método queue_declare() del canal que se ha creado previamente. La cola se llama “ejemplo”.

En los códigos de la máquina que hace de publicador se puede ver las siguientes dos líneas:

```
properties = pika.BasicProperties(headers={'timestamp_envio': str(time.time())})

channel.basic_publish(exchange='', routing_key='ejemplo', body=message, properties=properties)
```

La primera línea de código se crea un objeto de propiedades (*properties*) utilizando la clase BasicProperties de Pika. El objeto de propiedades se utiliza para adjuntar datos al mensaje, como en este caso se utiliza para agregar una cabecera con la marca de tiempo (timestamp_envio) del mensaje. En la línea de código, se utiliza la función time.time() para obtener la marca de tiempo actual en formato de cadena.

En la segunda línea de código se publica el mensaje en la cola de RabbitMQ utilizando el método basic_publish() del canal. El mensaje se publica en la cola llamada 'ejemplo' y se adjunta el cuerpo del mensaje (*message*). Además, el objeto de propiedades se proporciona como argumento de *properties* para incluir la cabecera con la marca de tiempo en el mensaje.

En el suscriptor se tiene el siguiente código:

```
timestamp_envio = float(properties.headers['timestamp_envio'])

delay_total = time.time() - timestamp_envio

tamanio_payload = len(body)
```

Estas tres líneas de código se utilizan para procesar un mensaje recibido de una cola de RabbitMQ. Se obtiene la marca de tiempo de envío del mensaje, se calcula el retraso total del mensaje y se obtiene el tamaño del *payload* del mensaje.

El código que se muestra a continuación sirve para escribir los resultados del procesamiento de mensajes en un archivo CSV utilizando la biblioteca csv de Python.

```
with open('resultados.csv', mode='a', newline='') as resultados_file:

    resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)

    resultados_writer.writerow([delay_total, tamanio_payload])
```

A continuación, se detallará las configuraciones particulares utilizadas para los niveles de calidad de servicio (QoS) 1 y 2.

Calidad de servicio QoS 1

La siguiente línea de código que está presente tanto en el publicador como en el suscriptor se utiliza para configurar el nivel de calidad de servicio (QoS) en una conexión AMQP utilizando la biblioteca pika. Al establecer prefetch_count=1 se indica que el consumidor procesará un mensaje a la vez de la cola de mensajes, lo que puede mejorar el rendimiento y la estabilidad de la aplicación. Esta configuración es adecuada para escenarios en los que es importante procesar los mensajes de forma secuencial y en orden.

```
channel.basic_qos(prefetch_count=1)
```

Calidad de servicio QoS 2

En la siguiente línea de código se configura un nivel de QoS 2 en el publicador, esta línea de código utiliza la biblioteca pika, se puede usar el método basic_qos() con los parámetros prefetch_count y global_qos establecidos en 0 y True, respectivamente. Además, se puede establecer el parámetro prefetch_size en 0 para deshabilitar la limitación de tamaño de mensajes.

```
channel.basic_qos(prefetch_size=0, prefetch_count=0, all_channels=True)
```

En el suscriptor la línea de código equivalente es:

```
channel.basic_qos(prefetch_size=0, prefetch_count=1, all_channels=True)
```

Al establecer prefetch_count=1, se indica que el suscriptor procesará un mensaje a la vez de la cola de mensajes. Al establecer all_channels=True, se indica que se desea aplicar esta configuración a todos los canales de la conexión, lo que garantiza que todos los mensajes se entreguen exactamente una vez, sin duplicados.

4.1.3.2. Prueba de latencia protocolo MQTT

Para el protocolo MQTT para el publicador y suscriptor se estableció al principio el siguiente código.

```
import paho.mqtt.client as mqtt

import time

client = mqtt.Client()
```

La primera línea de código importa la biblioteca Paho MQTT con el alias mqtt. La segunda línea de código importa el módulo time que se utiliza para medir el tiempo de latencia. La tercera línea de código inicializa un objeto de cliente MQTT utilizando el constructor de la clase Client, este objeto es la instancia del cliente que se utiliza para conectarse a un *broker* MQTT y publicar o suscribirse a un *topic*. En este caso, el objeto se inicializa sin parámetros, lo que significa que el objeto utilizará los valores por defecto para los parámetros de conexión y otros ajustes.

Para conectarse al *broker* MQTT se necesario insertar el siguiente código:

```
client.connect("192.168.0.184", 1883, 60)
```

Con esta función se toma como argumentos el *hostname* o la dirección IP del *broker* MQTT, el puerto en el que está escuchando el *broker* (generalmente el 1883 para conexiones no seguras) y el tiempo de espera máximo de la conexión en segundos.

En el publicador se debe incluir las siguientes dos líneas de código dentro de un ciclo for para enviar varios paquetes:

```
payload = "{}-{}".format(int(time.time() * 1000), "x" * (64 - len(str(int(time.time() * 1000)))))

client.publish("test/topic", payload, qos=0)
```

La primera línea de código crea un *payload* (carga útil) de mensaje con el formato “timestamp-x”, donde el timestamp es la hora actual multiplicada por 1000 para obtener la medición en ms y redondeado a un número entero, y "x" es un carácter que se repite hasta alcanzar una longitud de 64 caracteres.

La segunda línea de código publica el mensaje generado en el tema "test/topic" utilizando la conexión del cliente MQTT con el parámetro de QoS establecido en 0, lo que significa que el mensaje se entrega una vez sin garantía de recepción ni retransmisión en caso de pérdida o duplicación.

Si se configura la calidad de servicio QoS 1 significa que se garantiza que el mensaje se entregue al menos una vez al suscriptor, aunque pueden producirse duplicados. Mientras que con un QoS 2 se garantiza le entrega de mensajes exactamente una vez.

Para el suscriptor se utilizan las siguientes líneas de código en particular:

```
payload = msg.payload.decode("utf-8")

payload_time = int(payload.split("-")[0])

latency = time.time() * 1000 - payload_time

print("Latency: {}ms".format(latency))

writer.writerow([latency, len(payload)])

f.flush()
```

La primera línea de código decodifica el mensaje recibido en formato de bytes a una cadena de caracteres en formato utf-8 y lo guarda en la variable "payload". La segunda línea de código extrae la información de tiempo del mensaje que está en los primeros caracteres de la cadena, y lo convierte en un entero, el cual se guarda en la variable "payload_time". La tercera línea de código calcula la latencia de la comunicación restando el tiempo actual en milisegundos (obtenido con la función time.time()) del tiempo contenido en el mensaje "payload_time". La cuarta línea de codigo imprime el resultado de la latencia en la consola. Finalmente, la quinta y sexta línea de código se utilizan para escribir la latencia en un archivo CSV y guardar los cambios.

4.1.4. Ejecución de la prueba de perdida de paquetes

En esta prueba experimental se repitió la prueba de latencia para cada tamaño de paquete y calidad de servicio (QoS), y luego se calculó el porcentaje de paquetes perdidos de cada prueba. Este proceso se repitió 25 veces para reducir estos factores de incertidumbre, y así obtener resultados precisos y útiles.

El cálculo de los paquetes perdidos se puede calcular dividiendo el número total de mensajes que no se recibieron entre el número total de paquetes que no se enviaron, y multiplicando por 100 para obtener un porcentaje.

4.1.5. Ejecución de la prueba de rendimiento (*throughput*)

Finalmente se ejecutó la prueba de rendimiento, la cual es similar a la prueba de pérdida de paquetes, sin embargo, en este caso se dividió la cantidad de paquetes enviados exitosamente (de los 1000 que se manda por prueba) entre el tiempo total de latencia que tomo a los paquetes llegar al suscriptor.

4.2. Validación de Resultados

En esta sección se utilizaron gráficos de tipo diagrama de caja y bigotes para visualizar las diferencias y patrones en los resultados. Además, se presentaron los estadísticos descriptivos de cada prueba, incluyendo la media, mediana, desviación estándar, mínimo y máximo, para poder identificar valores extremos o atípicos.

Se analizaron las causas de los resultados y se identificarán las limitaciones del estudio y aprovechando los resultados obtenidos se comparará el desempeño de los protocolos AMQP y MQTT para sus diferentes niveles de calidad QoS.

También se contrastarán los resultados obtenidos en este trabajo con los resultados de otros tres estudios previos. Estos estudios llevaron a cabo pruebas experimentales distintas a las realizadas en este trabajo, por lo que se utilizarán como referencia para contextualizar y evaluar los resultados obtenidos en este trabajo. Luego se procederá a analizar e interpretar conjuntamente los resultados tanto del presente trabajo como de los otros estudios. Se reflexionará sobre las limitaciones de la comparación de resultados debido a las diferencias en los diseños experimentales.

4.2.1. Análisis estadístico descriptivo de los resultados obtenidos

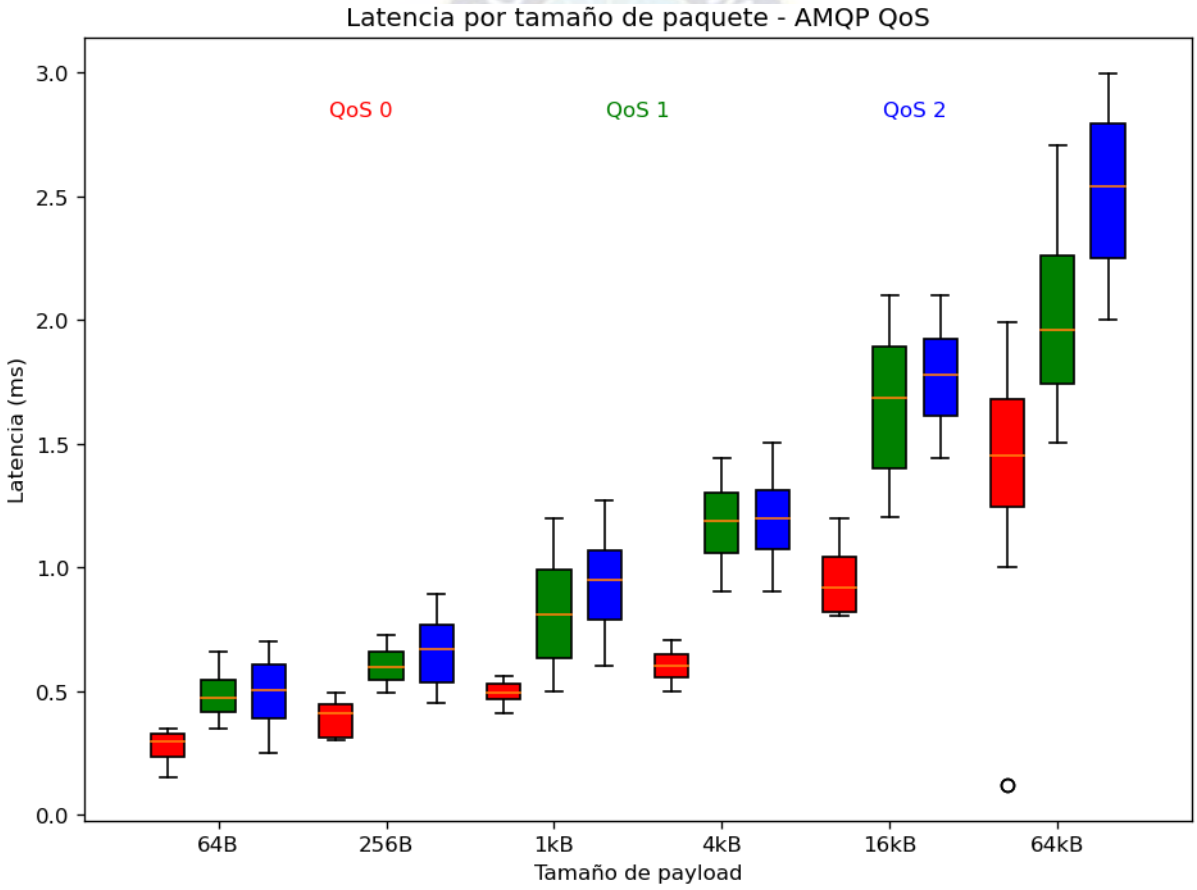
4.2.1.1. Análisis estadístico descriptivo de los resultados de latencia

En esta sección se hará un análisis descriptivo por protocolo de los resultados obtenidos en la prueba de latencia:

AMQP

Con los resultados obtenidos de la prueba del protocolo AMQP se elaboró una gráfica que describe el comportamiento de la latencia según el tamaño del *payload* (véase figura 37). En la gráfica se puede visualizar que la latencia más elevada se tiene para un nivel de calidad de servicio QoS 2, seguido del QoS 1 y luego por el QoS 0. Se puede observar también que mientras aumenta el *payload* la latencia aumenta. Para finalizar, es evidente mientras aumenta el tamaño del *payload*, la desviación estándar de la latencia aumenta, así por ejemplo para QoS 2 en 64kB se tiene una latencia de 2.5 ms. Un resumen más detallado de los resultados obtenidos para cada nivel de calidad QoS se puede observar en el Anexo 12.

Figura 37. Latencia para el protocolo AMQP

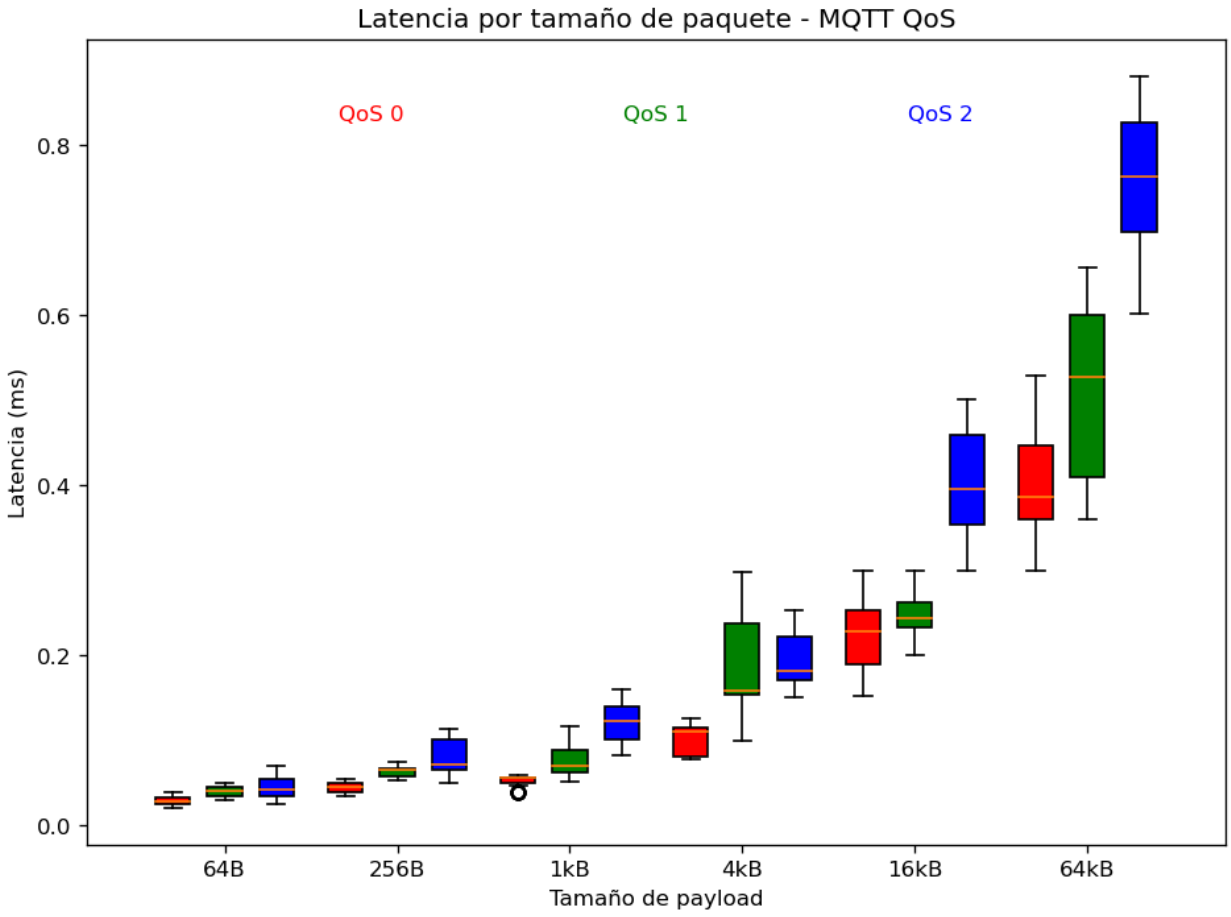


Fuente: Elaboración propia.

MQTT

En la figura 38 se resume los resultados obtenidos de la prueba de latencia para el protocolo MQTT. Los datos obtenidos reflejan que en general el nivel de calidad de servicio QoS 2 es el que tiene una mayor latencia. Para un *payload* de 4kB es evidente que los resultados obtenidos para un nivel de calidad QoS 1 muchas veces son mayores a los del nivel de calidad QoS 2, esto se puede deber a que muchas veces la red local familiar en la que se hicieron las pruebas puede que haya registrado un alto tráfico de información debido al uso de ciertos usuarios. Lo propio para este caso, el detalle más completo de los resultados obtenidos para cada nivel de calidad QoS se puede observar en el Anexo13.

Figura 38. Latencia para el protocolo MQTT

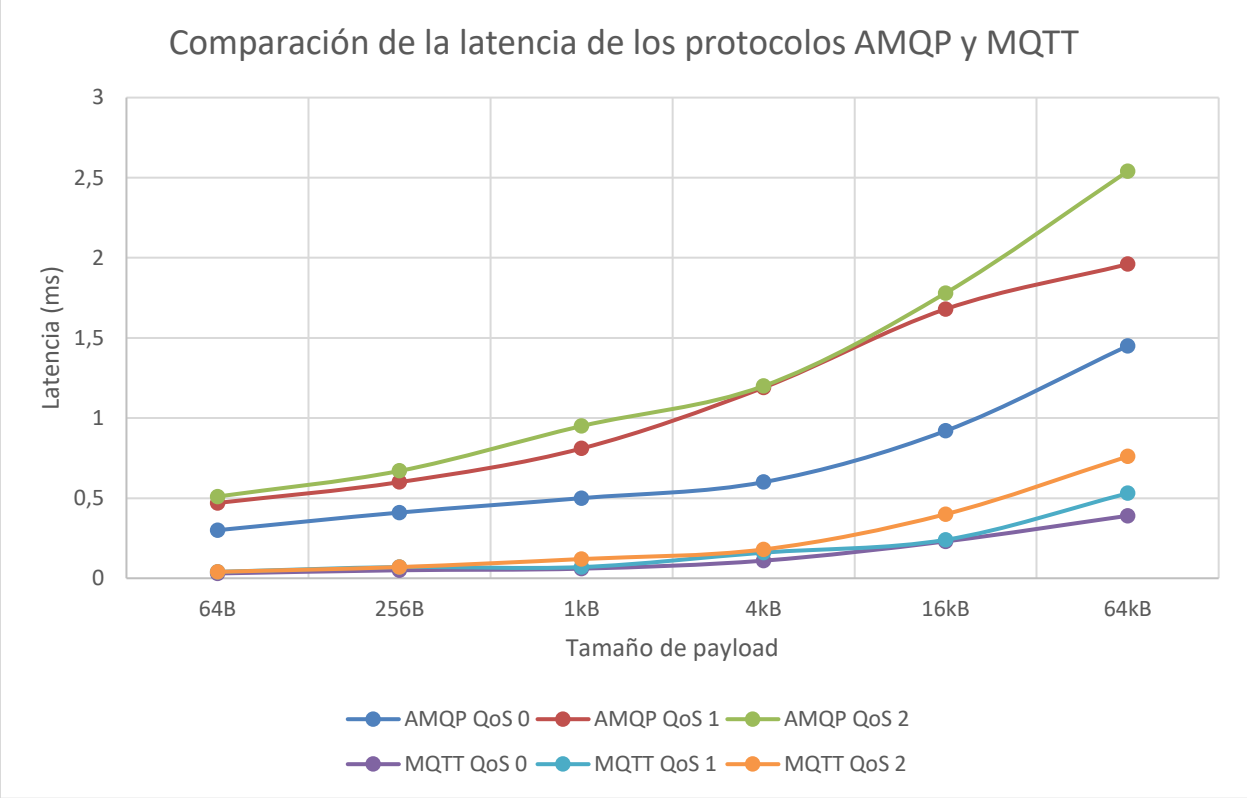


Fuente: Elaboración propia.

Aprovechando los resultados obtenidos de latencia para los protocolos AMQP y MQTT, podemos hacer comparaciones, para eso se hará uso de la siguiente grafica que muestra los valores de mediana de latencia.

Se puede observar de la figura 39 que el protocolo AMQP tiene una mayor latencia, además de que el nivel de calidad QoS 2 es el que tiene mayor latencia seguido de QoS 1 y para finalizar QoS 0.

Figura 39. Comparación de la latencia de los protocolos AMQP y MQTT

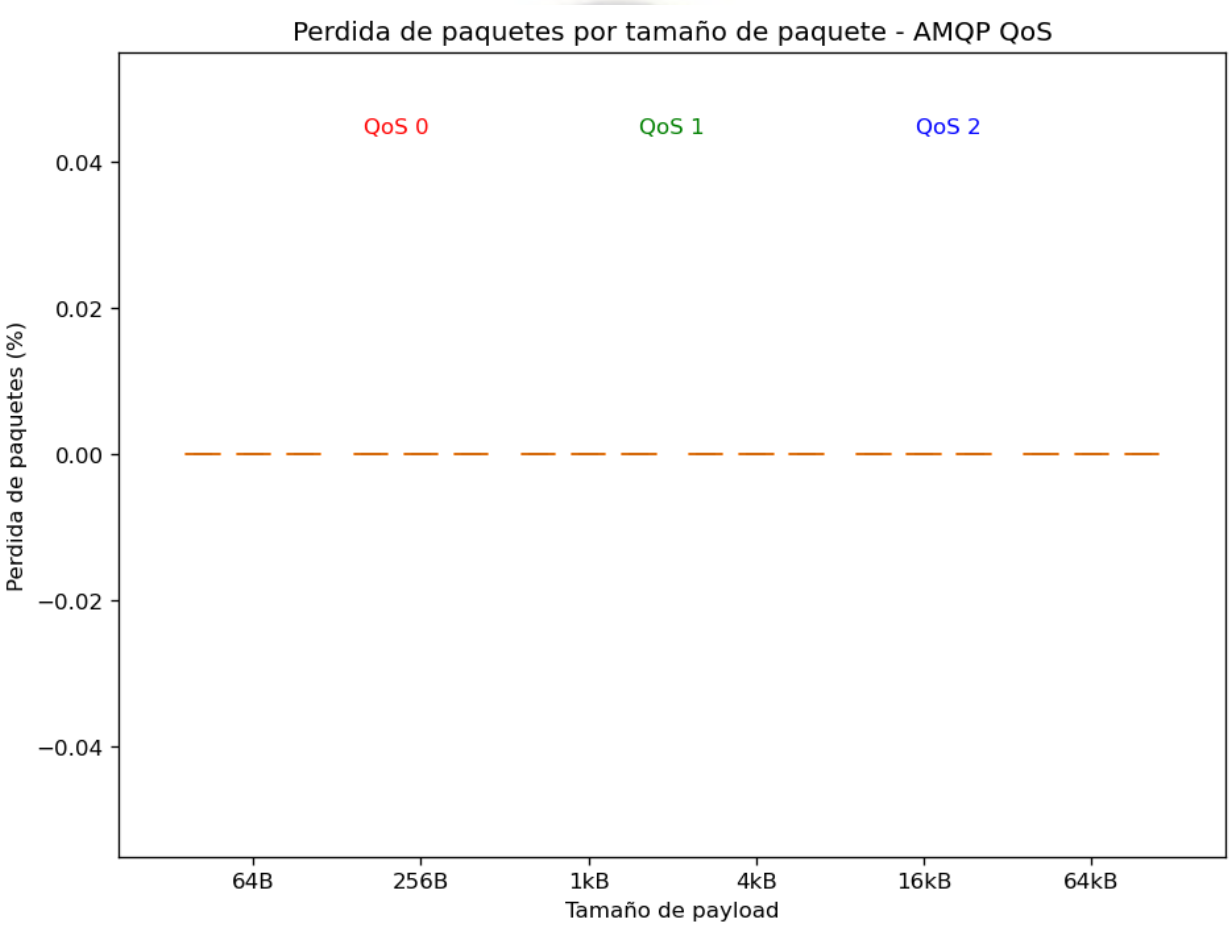


Fuente: Elaboración propia.

4.2.1.2. Análisis estadístico descriptivo de los resultados de paquetes perdidos AMQP

La figura 40 muestra los resultados obtenidos de la prueba de perdida de paquetes para el protocolo AMQP, se puede observar que el porcentaje de perdida de paquetes es nulo para los diferentes niveles de calidad de servicio QoS y tamaños de *payload*. Esto puede deberse a que en redes locales WiFi el protocolo AMQP no registra perdida de paquetes. En el Anexo 14 se muestra el detalle de los resultados obtenidos.

Figura 40. Perdida de paquetes protocolo AMQP

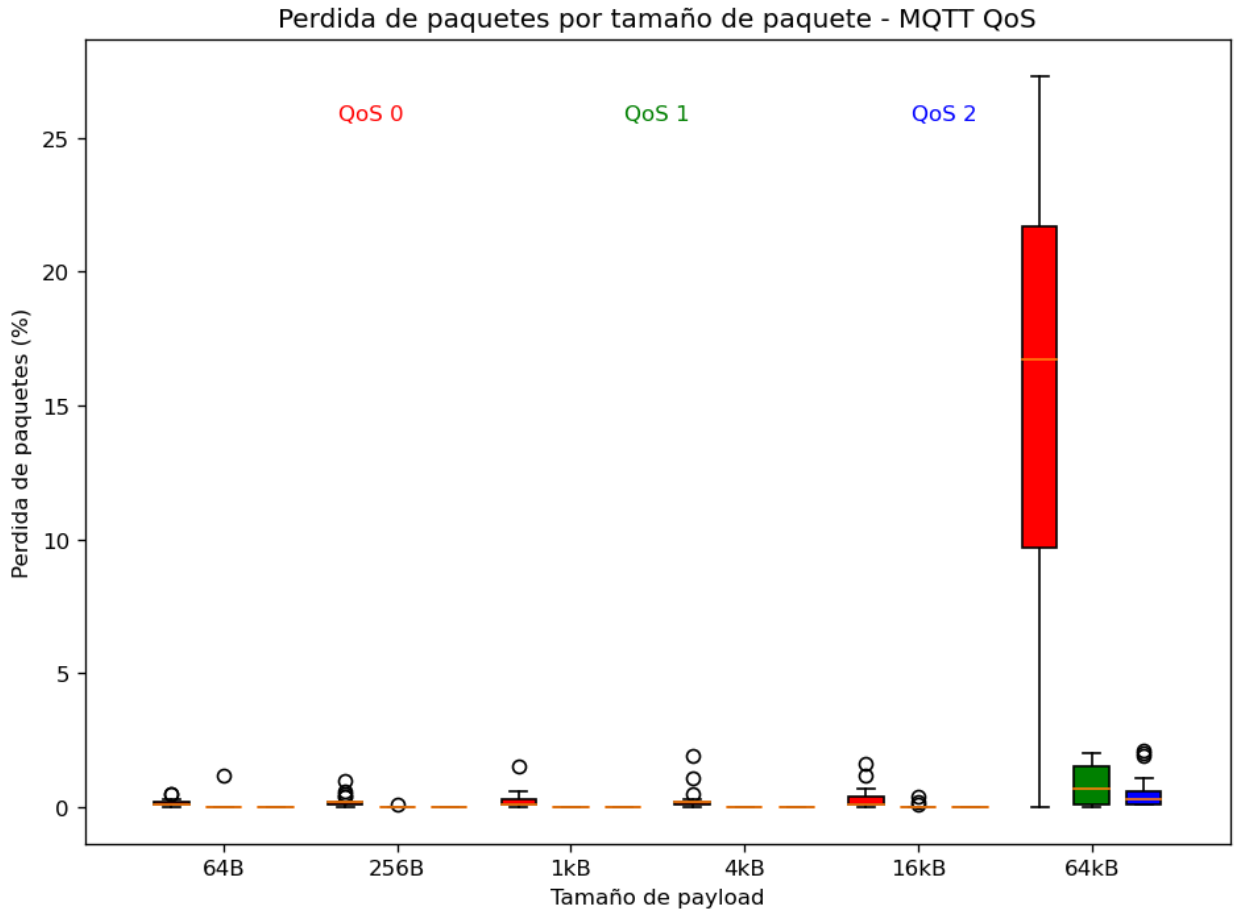


Fuente: Elaboración propia.

MQTT

En el caso del protocolo MQTT se puede observar que existen pérdidas de paquetes en los tres niveles de calidad. El porcentaje de pérdidas de paquetes es mayor en el nivel de calidad QoS 0, le sigue el nivel de calidad 1, para luego terminar con el nivel de calidad QoS 2 que tiene una perdida casi nula de paquetes. En la figura 41 también se puede observar que mientras más grande sea el *payload*, el porcentaje de perdida de paquetes aumenta. Es evidente que el nivel QoS 0 para un tamaño de 64kB registra una mayor pérdida de paquetes. En el Anexo 15 se puede observar más detalles de los resultados obtenidos.

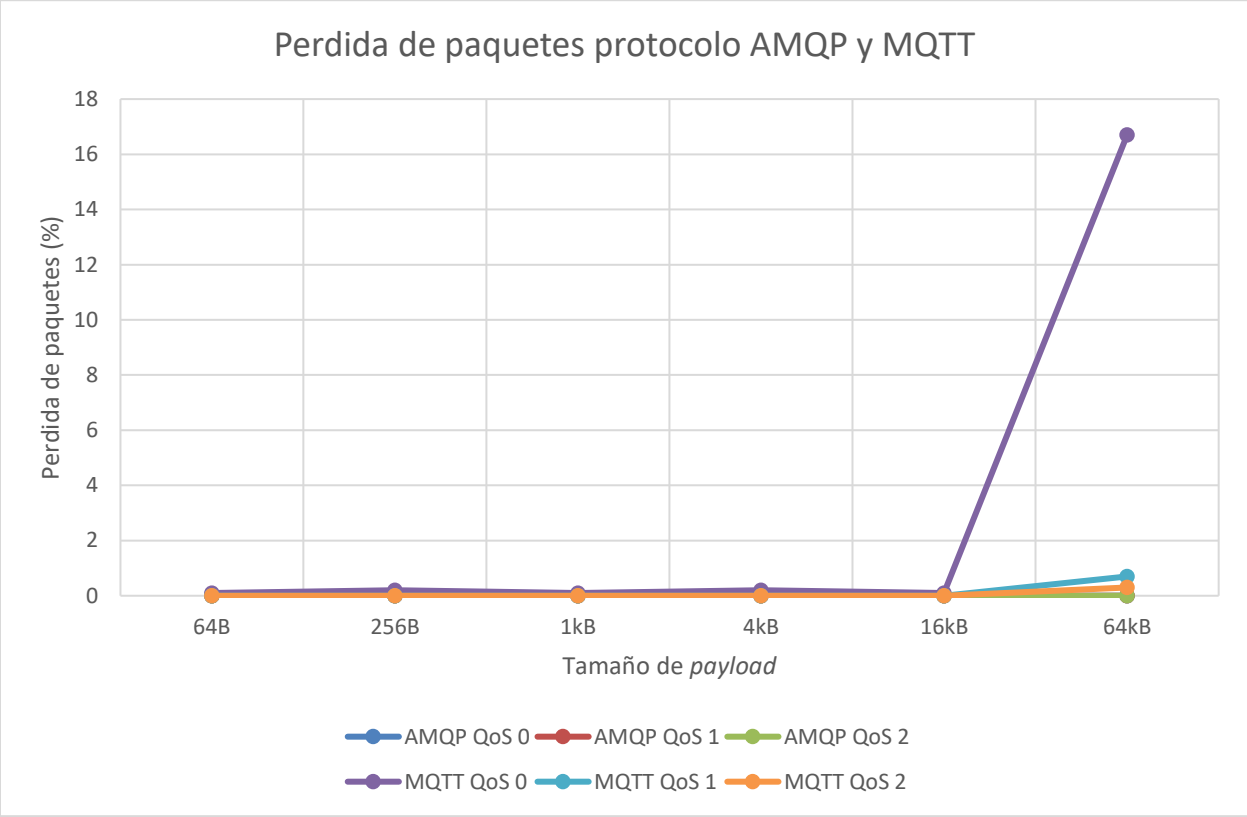
Figura 41. Perdida de paquetes protocolo MQTT



Fuente: Elaboración propia.

Comparando el porcentaje de perdida de paquetes de los protocolos AMQP y MQTT se puede concluir que el protocolo MQTT es el que registra mayor pérdida de paquetes y que las pérdidas de paquete aumentan a medida que el *payload* aumenta (véase figura 42).

Figura 42. Comparación de porcentaje de perdida de paquetes de los paquetes AMQP y MQTT



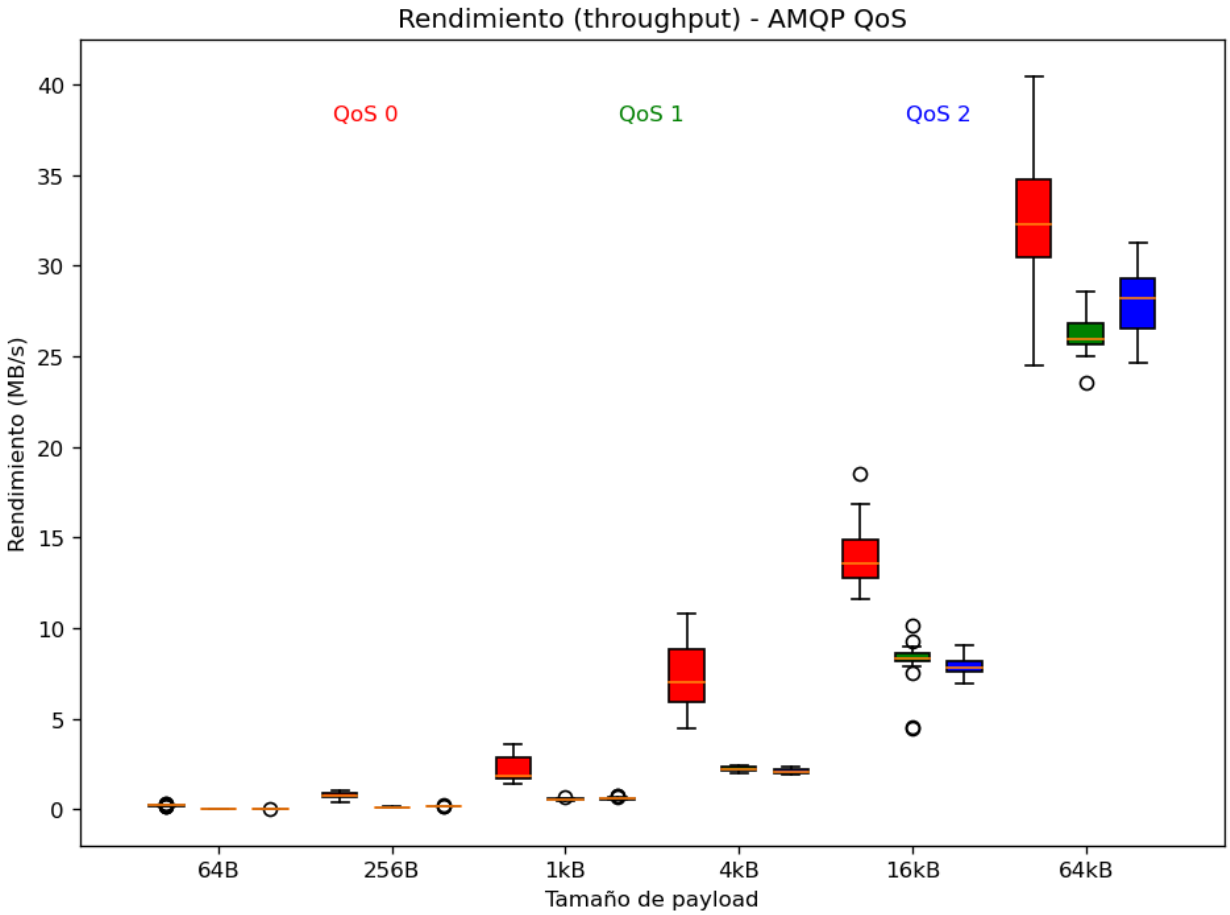
Fuente: Elaboración propia.

4.2.1.3. Análisis estadístico descriptivo de los resultados de rendimiento (*throughput*)

AMQP

Para el análisis del rendimiento se puede observar en la figura 43 que este aumenta a medida que aumenta el tamaño del *payload*. Se puede observar que el rendimiento en el nivel de calidad QoS 0 es el más alto, esto se debe mucho a que los paquetes en este nivel de calidad se envían más rápido que en los otros niveles. A pesar de esto, se puede observar que para el nivel de calidad QoS 1 para un tamaño de *payload* de 64kB su rendimiento es menor al nivel de QoS 2, esto puede deberse a que la red de prueba donde se encuentran las máquinas virtuales es domiciliaria y muchas veces existen usuarios que hacen uso intensivo de esta red, lo cual provoca alteraciones a los resultados esperados. Un reporte más completo de los resultados obtenidos se puede observar en el Anexo 16.

Figura 43. Rendimiento AMQP

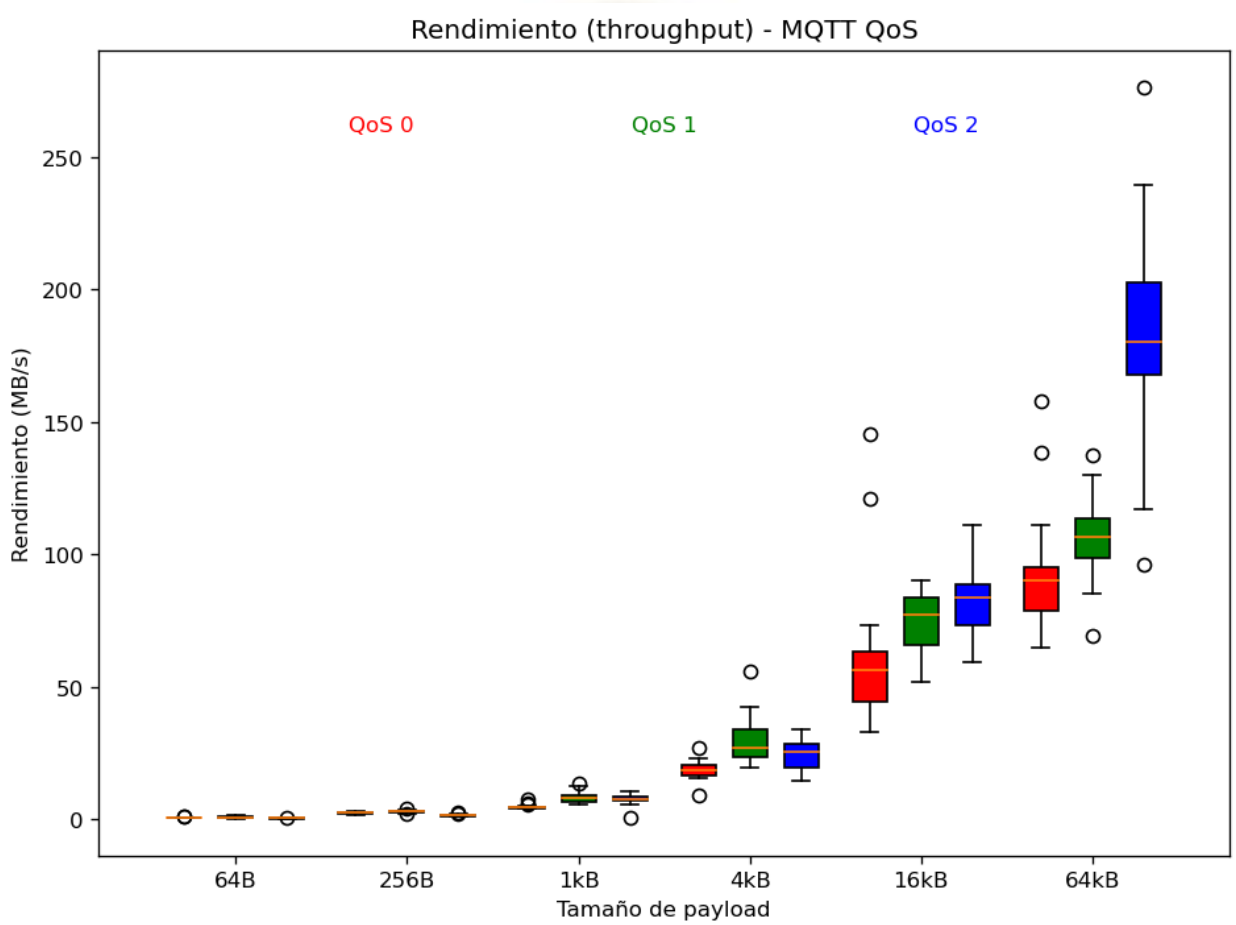


Fuente: Elaboración propia.

MQTT

Para el caso del protocolo MQTT se puede observar en la figura 44, que el protocolo tiene mejor rendimiento cuando trabaja en el nivel de calidad de servicio QoS 2, esto se debe a que en este nivel de calidad se transfiere más información en un periodo de tiempo determinado ya que casi no existe perdida de paquetes a pesar de que el tiempo de latencia de es estos paquetes es mayor. Para más detalle de los resultados obtenidos se puede revisar el Anexo 17.

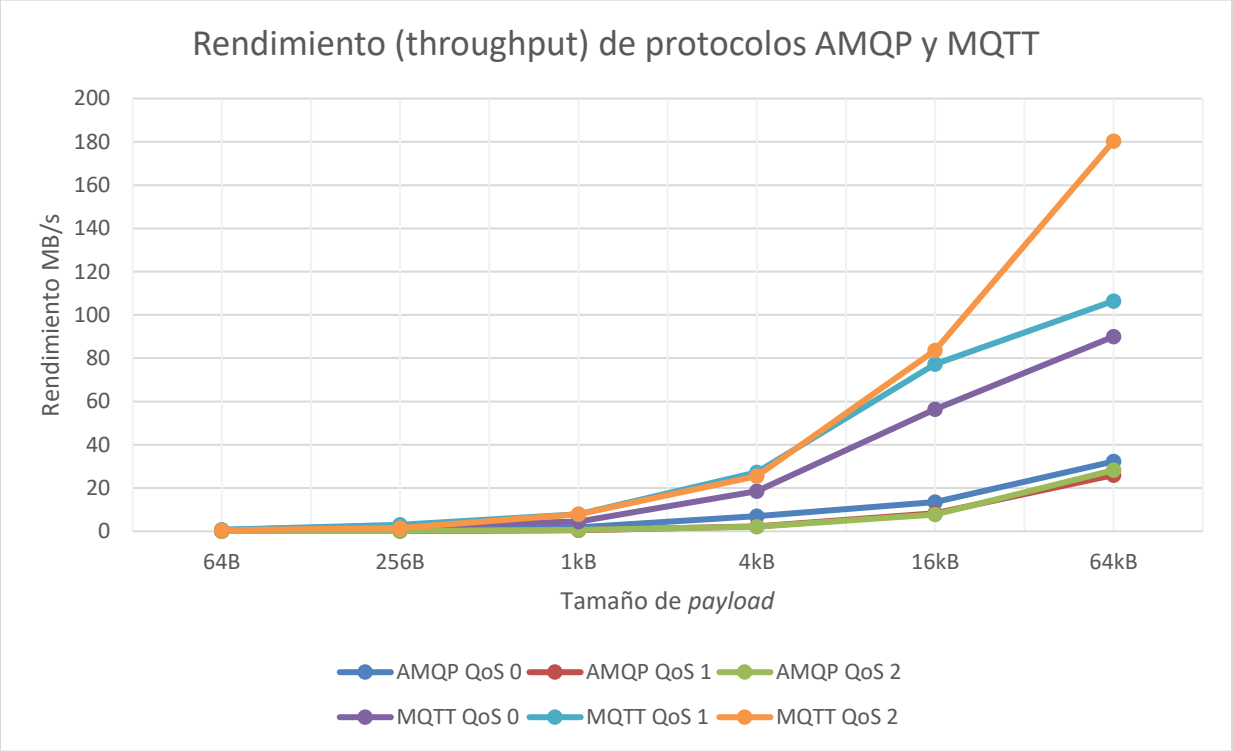
Figura 44. Rendimiento de protocolo MQTT



Fuente: Elaboración propia.

Para finalizar esta sección se comparará el rendimiento de los protocolos AMQP y MQTT, para eso se hizo una gráfica que muestra las medianas del rendimiento de los protocolos (véase figura 45). Es evidente que el protocolo MQTT tiene mejor rendimiento que el protocolo AMQP, además de que el protocolo MQTT para la calidad de servicio QoS 2 es más eficiente, seguido de la calidad de servicio QoS 1 y el nivel de calidad QoS 0.

Figura 45. Comparación de rendimiento de protocolos AMQP y MQTT



Fuente: Elaboración propia.

4.2.2. Discusión de los resultados con otros estudios similares

En esta sección se discutirá los resultados obtenidos en la presente tesis con los resultados obtenidos de otros tres trabajos. Se señalará las similitudes y diferencias clave entre los estudios, como las variables investigadas, las muestras utilizadas y los patrones encontrados.

Correlation Analysis of MQTT Loss and Delay According to QoS Level (Lee, Kim, Hong, & Ju, 2013)

En el artículo se analiza el proceso de transmisión de mensajes MQTT que consta de un cliente que publica con cable o en forma inalámbrica, un *broker* de intermediario y un cliente suscriptor. Al transmitir mensajes a través de 3 niveles de QoS con varios tamaños de carga útil, se capturaron paquetes para analizar la latencia y perdida de paquetes.

Los resultados obtenidos en el artículo (Lee, Kim, Hong, & Ju, 2013) indican que la latencia está asociada con la perdida de mensajes en diferentes *payloads*, si se aumenta el nivel de calidad (aumento de latencia) las pérdidas de paquetes se reducen, se aplica para todos los tamaños de *payload*. También se pudo evidenciar en el artículo la influencia del tamaño de *payload* en la latencia y la perdida de paquetes. Si se puede utilizar el protocolo MQTT utilizando el nivel de QoS adecuado para diferentes *payloads*, se podrá construir un entorno de red de servicio de notificación más efectivo.

En el Anexo 18 se puede ver las gráficas que resumen los resultados de este artículo (Lee, Kim, Hong, & Ju, 2013), estos resultados difieren de los que se obtuvieron en el presente trabajo debido a que influye mucho las condiciones de la red en la que se efectúan las pruebas. Sin embargo, se puede observar que, a pesar de la variación de resultados, las tendencias de latencia y perdida de paquetes se parecen. Es decir, para el caso de la latencia el nivel de calidad QoS 2tiene mayor *delay*, seguido del nivel de calidad Qos 1 y el nivel de calidad QoS 0, todo esto para diferentes *payloads*. Mientras que para la perdida de paquetes el nivel de calidad QoS 0 registra más perdidas, seguido del nivel de calidad QoS 1 y el nivel de calidad QoS 2.

A Real Time Demonstrative Analysis of Lightweight Payload Encryption in Resource Constrained Devices Based on MQTT (Menyah, 2017)

Este trabajo presenta un análisis experimental del efecto de la encriptación de datos en el rendimiento de la red usando el protocolo MQTT para dispositivos IoT con recursos limitados. En el trabajo se utiliza un algoritmo de encriptación ligero AES-128 para cifrar los datos enviados por un dispositivo Arduino a través de un servidor intermediario que usa el software Mosquitto a otro dispositivo Raspberry Pi, también se utilizó un enrutador inalámbrico TP-Link 150Mbps AP/Client con soporte LAN cableado.

El autor mide la latencia, el rendimiento, la perdida de mensajes y el tiempo de procesamiento de la encriptación y la desencriptación para diferentes niveles de calidad de servicio QoS y tamaños de carga útil.

Los resultados del experimento muestran que la encriptación tiene un impacto negativo en el rendimiento de la red, aumentando la latencia, la pérdida de mensajes y el tiempo de procesamiento, y disminuyendo el rendimiento. Sin embargo, el autor argumenta que este impacto es aceptable considerando los beneficios de seguridad que ofrece la encriptación. El autor también observa que el nivel de QoS influye en el rendimiento de la red, siendo el QoS 0 el más rápido y el QoS 2 el más lento. El autor concluye que la encriptación es una solución viable para mejorar la seguridad de los dispositivos IoT con recursos limitados y recomienda usar el nivel de QoS adecuado según las necesidades de cada aplicación.

En el Anexo 19 se muestra los resultados que se obtuvieron en la tesis, es evidente que los valores difieren de los que se obtuvieron en la presente tesis y es que las redes donde se hicieron las pruebas tienen diferentes características. Sin embargo, el comportamiento de la latencia, la perdida de mensajes y el tiempo de procesamiento se parece mucho para cada nivel de calidad de servicio QoS con respecto al protocolo MQTT.

Performance Evaluation of Different Raspberry pi Models as MQTT Servers and Clients (N. Ford, Gamess, & Ogden, 2022)

Este articulo nos muestra una evaluación empírica del rendimiento del protocolo MQTT en dispositivos Raspberry Pi. Los autores usaron dos escenarios de prueba y dos herramientas de medición para analizar el tiempo de transmisión y el rendimiento de los mensajes publicados por los clientes MQTT a través de un intermediario (*broker*).

En el primer escenario, sin un ataque DoS, encontraron que el tiempo de transmisión de los mensajes MQTT depende principalmente del tamaño del mensaje, el nivel de QoS y el nivel de seguridad. El ancho de banda WiFi y el hardware utilizado tuvieron un impacto menor. Los resultados mostraron que el tiempo de transmisión aumenta con el tamaño del mensaje, el nivel de QoS y el nivel de seguridad. El tiempo de transmisión más bajo se obtuvo con mensajes de 100 bytes, QoS 0 y MQTT sin TLS. El tiempo de transmisión más alto se obtuvo con mensajes de 25.000 bytes, QoS 2 y MQTT con TLS.

En el segundo escenario, con un ataque DoS, encontraron que los dispositivos Raspberry Pi son vulnerables a un ataque TCP SYN *flood*, que puede causar una pérdida significativa de paquetes y una disminución del rendimiento. Los resultados mostraron que el tiempo de transmisión se incrementa drásticamente cuando se lanza el ataque, especialmente con mensajes grandes, QoS 2 y MQTT con TLS. El dispositivo más afectado fue el RPi Zero W, que sufrió una pérdida de paquetes del 100% en algunos casos. El dispositivo menos afectado fue el RPi 4B, que logró mantener una pérdida de paquetes inferior al 10% en la mayoría de los casos.

En cuanto al rendimiento, encontraron que el RPi 4B superó significativamente al resto de los dispositivos en términos de *throughput* y velocidad de procesamiento. El RPi Zero W fue el dispositivo con peor rendimiento en ambos aspectos. También observaron que el *throughput* y la velocidad de procesamiento disminuyen con el nivel de QoS y el nivel de seguridad.

Este articulo nos muestra un estudio completo del desempeño del protocolo MQTT y puede servir para orientar a los desarrolladores cómo funciona el protocolo MQTT, mostrando el rendimiento que puede esperarse según las diferentes opciones de Raspberry Pi. Viendo los resultados que se obtuvieron en el artículo también podemos confirmar que los resultados no son iguales debido a que la red donde se hicieron las pruebas difiere de la red en donde se hicieron las pruebas de la presente tesis. En el

Anexo 20 se muestra los resultados de las pruebas similares que se hicieron en el artículo con respecto a la tesis.

5. CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

En la presente tesis se logró satisfactoriamente documentar el estado del arte de los protocolos para la capa de aplicación utilizados en la IoT para entender sus características y especificaciones.

Posteriormente, se desarrolló una técnica de evaluación de desempeño de protocolos para la capa de aplicación utilizados en la IoT, con el fin de medir y comparar el desempeño de estos protocolos en términos de latencia, pérdida de paquetes y rendimiento. Para ello, se diseñaron e implementaron pruebas experimentales que simularon el envío y recepción de mensajes entre dispositivos IoT utilizando los protocolos AMQP y MQTT, la red de prueba fue una local domiciliaria WiFi.

Los resultados obtenidos se analizaron mediante estadística descriptiva y se compararon entre sí para determinar el desempeño relativo de cada protocolo. Los principales hallazgos del estudio fueron los siguientes:

- En términos de latencia se pudo observar que para ambos protocolos en el nivel de calidad QoS 2 se tiene mayor latencia, le sigue el nivel de calidad QoS 1 y el nivel de calidad QoS 0, respectivamente. El nivel de calidad QoS 0 tiene menor latencia en comparación con los otros niveles porque no requiere confirmaciones de entrega de mensajes entre el cliente y el broker. Comparando la latencia de los protocolos AMQP y MQTT se puede concluir que el protocolo AMQP tiene mayor latencia que el protocolo MQTT.
- En términos de pérdida de paquetes se pudo observar que el protocolo AMQP no presento perdidas de paquete en ninguno de sus tres niveles de calidad. Mientras que el protocolo MQTT presenta un 16.7% de perdida de paquetes en el nivel de calidad QoS 0, y que las pérdidas de paquetes se reducen en el nivel de calidad QoS 1 y aún más en el nivel de calidad QoS 2. Comparando la tasa de perdida de paquetes de los dos protocolos se puede concluir que el protocolo MQTT tiene una mayor pérdida de paquetes que AMQP.
- En términos de rendimiento (*throughput*) se pudo observar que el protocolo MQTT tiene mejor rendimiento en el nivel de calidad QoS 2, le sigue el nivel de calidad QoS 1 y el nivel de calidad QoS 0, respectivamente. Mientras que en el protocolo AMQP el nivel de calidad QoS 0 tiene mejor rendimiento, le sigue el nivel de calidad 1 y el nivel de calidad QoS 2, respectivamente. La diferencia del comportamiento del rendimiento del protocolo AMQP con respecto al protocolo MQTT se debe a que en el protocolo AMQP el rendimiento solo es afectado por el tiempo que tarda en transmitirse la información y a la nula perdida de paquetes. Comparando el rendimiento de los protocolos AMQP y MQTT se puede concluir

que el rendimiento del protocolo del protocolo MQTT es mucho mayor que el del protocolo AMQP.

En conclusión, el estudio demostró que el protocolo MQTT presenta un mejor desempeño que el protocolo AMQP en términos de latencia y rendimiento (*throughput*), y que el protocolo AMQP presenta un mejor desempeño en termino de pérdidas de paquete. Estos hallazgos son útiles para guiar la selección y evaluación de desempeño de los protocolos para la capa de aplicación utilizados en la IoT.

5.2. Recomendaciones

A pesar de que se lograron los objetivos planteados se recomienda para futuros trabajos de investigación los siguientes puntos:

- Explorar el impacto de otros factores que pueden afectar el desempeño de los protocolos, como el consumo de la energía, la congestión de la red y la privacidad de los datos.
- Extender la técnica de evaluación a otras capas del modelo OSI para obtener una visión más completa del rendimiento de los sistemas IoT.
- Sería interesante evaluar la técnica en diferentes entornos y escenarios, tales como redes industriales, redes celulares, entre otros, para obtener una visión más completa y diversa del desempeño de los protocolos en la IoT.

6. REFERENCIAS BIBLIOGRÁFICAS

Bibliografía

¿Qué es IoT? (s.f.). Recuperado el 04 de julio de 2023, de Amazon Web Services: https://aws.amazon.com/es/what-is/iot/

Advanced Message Queuing Protocol. (2 de diciembre de 2021). Recuperado el 19 de octubre de 2022, de Wikipedia: https://es.wikipedia.org/w/index.php?title=Advanced_Message_Queuing_Protocol&oldid=140107043

Akilli, M. (2018). *Analysis of Transformation Capabilities between Communication Types of Cloud Application Components.* University of Stuttgart, Stuttgart, Alemania. Recuperado el 19 de octubre de 2022

AMQP 0-9-1 Model Explained. (s.f.-d). Recuperado el 28 de octubre de 2022, de RabbitMQ: https://www.rabbitmq.com/tutorials/amqp-concepts.html

AMQP: Introducing the Advanced Message Queuing Protocol. (3 de mayo de 2019). Recuperado el 20 de octubre de 2022, de IONOS Digital Guide: https://www.ionos.com/digitalguide/websites/web-development/advanced-message-queuing-protocol-amqp/

Barbosa, R. (27 de noviembre de 2015). *Modelo OSI: la guía definitiva del Modelo OSI !!!* Recuperado el 10 de octubre de 2022, de seaccna: https://seaccna.com/modelo-osi-guia-definitiva/

Bassi, A. (22 de julio de 2021). *Introducción al protocolo AMQP*. Recuperado el 28 de octubre de 2022, de Goto IoT!: https://www.gotoiot.com/pages/articles/amqp_intro/index.html

Bassi, A. (1 de julio de 2021). *Introducción al protocolo CoAP*. Recuperado el 20 de julio de 2022, de https://www.gotoiot.com/pages/articles/coap_intro/index.html

BytesofGigabytes. (27 de enero de 2019). *MQTT Packet Structure*. Recuperado el 9 de marzo de 2022, de https://bytesofgigabytes.com/mqtt/mqtt-protocol-packet-structure/

Fernández, M., & Gabriel , T. (2016). HTTP/2 Un nuevo protocolo para la web. *(Tesis de maestría).* Universidad Nacional de Lujan, Argentina.

García Davis, E. J. (2019). *Contribution to the Publish/Subscribe Communication Model for the Development of Ubiquitous Services in Wireless Sensor Networks.*

Universidad Politecnica de Cataluña, Barcelona. Recuperado el 12 de octubre de 2022

Gartner. (s.f.-a). *Definition of Internet of Things (IoT)*. Recuperado el 13 de agosto de 2021, de https://www.gartner.com/en/information-technology/glossary/internet-of-things

Gimeno Gimenez, X. (2013). Desarrollo y Estudio del Protocolo Observe para CoAP. *(Tesis de Grado).* Universidad Politecnica de Cataluña, España.

Hari, S. (21 de febrero de 2018). *Asynchronous Web Scraping*. Obtenido de https://santhoshhari.github.io/: https://santhoshhari.github.io/2018/02/21/asynchronous-web-scraping.html

Hervas Parra, C. (2018). Análisis de rendimiento de protocolos de Publicación/Subscripción en comunicación con una Red de Sensores Inalámbricos Zigbee. *(Tesis de Maestría).* Universidad Nacional de la Plata, Argentina. Recuperado el 20 de julio de 2022

Identificador de recursos uniforme. (16 de julio de 2022). Recuperado el 20 de julio de 2022, de Wikipedia: https://es.wikipedia.org/w/index.php?title=Identificador_de_recursos_uniforme&oldid=144803038

Internet of things. (7 de agosto de 2021). Obtenido de Wikipedia: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=1037627970

IoT and non-IoT connections worldwide 2010-2025. (8 de marzo de 2021). Recuperado el 5 de octubre de 2021, de Statista: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

Johansson, L. (23 de septiembre de 2019). *Part 1: RabbitMQ for beginners - What is RabbitMQ?* Recuperado el 28 de octubre de 2022, de CloudAMQP: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

Kinsta. (3 de septiembre de 2020). *kinsta.com*. Recuperado el 16 de julio de 2022, de https://kinsta.com/es/aprender/que-es-http2/#http2_feature_upgrades

Lee, S., Kim, H., Hong, D.-k., & Ju, H. (2013). Correlation Analysis of MQTT Loss and Delay . *The International Conference on Information Networking*, 714-717.

Londoño Delgado, D. F. (2020). HTTP/2 para internet de las cosas: analísis, evaluación y adaptación del control de flujo. *(Tesis de maestría).* Universidad de Chile, Santiago, Chile.

Maceda Dal-re, P. (2016). SNMP VS COAP EN SISTEMAS DE. *(Tesis de grado).* Universidad Pontificia Comillas, Madrid. Recuperado el 20 de julio de 2022

Marín Pérez, J. (2017). Análisis de prestaciones del protocolo HTTP2. *(Tesis de maestría).* Universidad Politectnica de Valencia, Valencia, España.

Marreros Guzman, J. (19 de enero de 2021). *¿Qué es HTTP/2 y qué ventajas tiene?* Recuperado el 17 de julio de 2022, de https://www.webempresa.com/blog/que-es-http2.html

Menyah, N. (2017). A real time demonstrative analysis of lightweight payload encryption in resource constrained devices based on MQTT. *Tesis para maestria.* Sakarya University, Serdivan.

Messaging patterns. (s.f.-b). Recuperado el 13 de octubre de 2022, de IBM: https://www.ibm.com/docs/en/wip-mg/2.0.0?topic=planning-messaging-patterns

Modelo OSI. (29 de septiembre de 2022). Recuperado el 10 de octubre de 2022, de Wikipedia: https://es.wikipedia.org/w/index.php?title=Modelo_OSI&oldid=146261637

MQTT. (3 de marzo de 2022). Recuperado el 7 de marzo de 2022, de Wikipedia: https://en.wikipedia.org/w/index.php?title=MQTT&oldid=1075037161

N. Ford, T., Gamess, E., & Ogden, C. (Marzo de 2022). Performance Evaluation of Different Raspberry Pi Models as MQTT Servers and Clients. *International Journal of Computer Networks and Communications, 14*(2), 1-18. doi:10.5121/ijcnc.2022.14201

OASIS. (s.f.-c). *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0*. Recuperado el 20 de octubre de 2022, de http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html

Pollard, B. (2019). *HTTP/2 in Action.* Shelter Island, Estados Unidos: Manning Publications Co.

Portas, L. (2021). Evaluación de la Seguridad de Sistemas IoT Basados en MQTT. *Tesis de maestría.* Universidad de Vigo, Coruña.

Publish–subscribe pattern. (29 de junio de 2022). Recuperado el 13 de octubre de 2022, de Wikipedia:

https://en.wikipedia.org/w/index.php?title=Publish%E2%80%93subscribe_pattern&oldid=1095582298

Request–response. (23 de septiembre de 2022). Recuperado el 13 de octubre de 2022, de Wikipedia: https://en.wikipedia.org/w/index.php?title=Request%E2%80%93response&oldid=1111967471

Salas, D. (2018). Security analysis for MQTT in Internet of Things. *(Tesis de maestría).* KTH Royal Institute of Technology, Estocolmo, Suecia.

Shelby, Z., Hartke, K., & Bormann, C. (20 de diciembre de 2018). *The Constrained Application Protocol (CoAP)*. Recuperado el 20 de julio de 2022, de https://datatracker.ietf.org/doc/rfc7252/

Solace. (04 de marzo de 2022). *2 MQTT Control Packet format*. Recuperado el 03 de marzo de 2022, de https://docs.solace.com/MQTT-311-Prtl-Conformance-Spec/MQTT%20Control%20Packet%20format.htm

7. ANEXOS

7.1. Anexo 1

En la tabla 5 se observan las opciones definidas en el protocolo CoAP son:

Tabla 5. Opciones del protocolo CoAP

Option Number	Option
1	Content-Type
3	Uri-Host
4	ETag
5	If-None-Match
7	Uri-Port
8	Location-Path
11	Uri-Path
12	Content-Format
14	Max-Age
15	Uri-Query
16	Accept
20	Location-Query
35	Proxy-Uri
39	Proxy-Scheme

Fuente: Tomado de Shelby et al. (2018).

7.2. Anexo 2

El tipo de mensaje representa el tipo de solicitud de conexión, este se define mediante un valor y tiene una longitud de cuatro bits (BytesofGigabytes, 2019) (véase tabla 6).

Tabla 6. Tipos de mensaje MQTT

Nombre	Valor	Dirección de flujo	Descripción
Reserved	0	Prohibido	Reservado
CONNECT	1	Cliente a servidor	Solicitud del cliente para conectarse al servidor
CONNACK	2	Servidor a cliente	Acuse de recibo de conexión
PUBLISH	3	Cliente a servidor o servidor a cliente	Publicar mensaje
PUBACK	4	Cliente a servidor o servidor a cliente	Publicar reconocimiento
PUBREC	5	Cliente a servidor o servidor a cliente	Publicación recibida (entrega asegurada parte 1)
PUBREL	6	Cliente a servidor o servidor a cliente	Publicar comunicado (entrega asegurada parte 2)
PUBCOMP	7	Cliente a servidor o servidor a cliente	Publicación completa (entrega asegurada parte 3)
SUBSCRIBE	8	Cliente a servidor	Solicitud de suscripción del cliente
SUBACK	9	Servidor a cliente	Confirmación de suscripción
UNSUBSCRIBE	10	Cliente a servidor	Solicitud de cancelación de suscripción
UNSUBACK	11	Servidor a cliente	Confirmación de cancelación de suscripción
PINGREQ	12	Cliente a servidor	Solicitud PING
PINGRESP	13	Servidor a cliente	Respuesta PING
DISCONNECT	14	Cliente a servidor	El cliente se está desconectando
Reserved	15	Prohibido	Reservado

Fuente: Adaptado de Solace (2022).

7.3. Anexo 3

Código empleado para la prueba de latencia para el protocolo AMQP para calidad de servicio QoS 0.

Equipo publicador

```
import pika

import time

# Conexión al broker

connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185', 5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo')

# Publicar los mensajes

for i in range(1000):

    # Crear el mensaje con una marca de tiempo de envío

    # Este payload se debe variar para los siguientes valores: 64B, 256B, 1kB, 4kB, 16kB y 64kB

    message = "x" * 64

    properties = pika.BasicProperties(headers={'timestamp_envio': str(time.time())})

    # Publicar el mensaje

    channel.basic_publish(exchange='', routing_key='ejemplo', body=message, properties=properties)

    time.sleep(1)

# Cerrar la conexión

connection.close()
```

Equipo suscriptor

```
import pika

import time

import csv

# Crear archivo CSV para registrar resultados

with open('resultados.csv', mode='w', newline='') as resultados_file:

    resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    resultados_writer.writerow(['delay_total', 'tamanio_payload'])

# Suscripción al mensaje

def callback(ch, method, properties, body):

    # Obtener la marca de tiempo de envío del mensaje

    timestamp_envio = float(properties.headers['timestamp_envio'])

    # Calcular el delay total

    delay_total = time.time() - timestamp_envio

    # Obtener el tamaño del payload

    tamanio_payload = len(body)

    # Guardar los resultados en el archivo CSV

    with open('resultados.csv', mode='a', newline='') as resultados_file:

        resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

        resultados_writer.writerow([delay_total, tamanio_payload])

    print("Latency: {}ms".format(delay_total))

    #print(" [x] Recibido %r" % body)

# Conexión al broker
```

```
connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185',
5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo')

# Suscribirse a la cola

channel.basic_consume(queue='ejemplo', on_message_callback=callback,
auto_ack=True)

# Escuchar por mensajes entrantes

print(' [*] Esperando mensajes. Presione CTRL+C para salir')

channel.start_consuming()

# Cerrar la conexión

connection.close()
```

7.4. Anexo 4

Código empleado para la prueba de latencia para el protocolo AMQP para calidad de servicio QoS 1.

Equipo publicador

```
import pika

import time

# Conexión al broker

connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185', 5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo_qos1', durable=True)

# Configurar QoS

channel.basic_qos(prefetch_count=1)

# Publicar los mensajes

for i in range(1000):

    # Crear el mensaje con una marca de tiempo de envío

    # Este payload se debe variar para los siguientes valores: 64B, 256B, 1kB, 4kB, 16kB y 64kB

    message = "x" * 64

    properties = pika.BasicProperties(headers={'timestamp_envio': str(time.time())})

    # Publicar el mensaje

    channel.basic_publish(exchange='', routing_key='ejemplo_qos1', body=message, properties=properties)

    time.sleep(1)

# Cerrar la conexión

connection.close()
```

Equipo suscriptor

```
import pika

import time

import csv

# Crear archivo CSV para registrar resultados

with open('resultados.csv', mode='w', newline='') as resultados_file:

    resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    resultados_writer.writerow(['delay_total', 'tamanio_payload'])

# Suscripción al mensaje

def callback(ch, method, properties, body):

    # Obtener la marca de tiempo de envío del mensaje

    timestamp_envio = float(properties.headers['timestamp_envio'])

    # Calcular el delay total

    delay_total = time.time() - timestamp_envio

    # Obtener el tamaño del payload

    tamanio_payload = len(body)

    # Guardar los resultados en el archivo CSV

    with open('resultados.csv', mode='a', newline='') as resultados_file:

        resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

        resultados_writer.writerow([delay_total, tamanio_payload])

    print("Latency: {}ms".format(delay_total))

    # Confirmar el mensaje recibido

    ch.basic_ack(delivery_tag=method.delivery_tag)

# Conexión al broker
```

```
connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185',
5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo_qos1', durable=True)

# Configurar QoS

channel.basic_qos(prefetch_count=1)

# Suscribirse a la cola

channel.basic_consume(queue='ejemplo_qos1', on_message_callback=callback)

# Escuchar por mensajes entrantes

print(' [*] Esperando mensajes. Presione CTRL+C para salir')

channel.start_consuming()

# Cerrar la conexión

connection.close()
```

7.5. Anexo 5

Código empleado para la prueba de latencia para el protocolo AMQP para calidad de servicio QoS 2.

Equipo publicador

```
import pika

import time

# Conexión al broker

connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185', 5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo_qos2', durable=True)

# Configurar QoS

channel.basic_qos(prefetch_size=0, prefetch_count=0, all_channels=True)

# Publicar los mensajes

for i in range(1000):

    # Crear el mensaje con una marca de tiempo de envío

    # Este payload se debe variar para los siguientes valores: 64B, 256B, 1kB, 4kB, 16kB y 64kB

    message = "x" * 64

    properties = pika.BasicProperties(headers={'timestamp_envio': str(time.time())}, delivery_mode=2)

    # Publicar el mensaje

    channel.basic_publish(exchange='', routing_key='ejemplo_qos2', body=message, properties=properties)

    print("Publicado mensaje {}".format(i))

# Cerrar la conexión
```

```
connection.close()
```

Equipo suscriptor

```
import pika

import time

import csv

# Crear archivo CSV para registrar resultados

with open('resultados.csv', mode='w', newline='') as resultados_file:

    resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    resultados_writer.writerow(['delay_total', 'tamanio_payload'])

# Suscripción al mensaje

def callback(ch, method, properties, body):

    # Obtener la marca de tiempo de envío del mensaje

    timestamp_envio = float(properties.headers['timestamp_envio'])

    # Calcular el delay total

    delay_total = time.time() - timestamp_envio

    # Obtener el tamaño del payload

    tamanio_payload = len(body)

    # Guardar los resultados en el archivo CSV

    with open('resultados.csv', mode='a', newline='') as resultados_file:

        resultados_writer = csv.writer(resultados_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

        resultados_writer.writerow([delay_total, tamanio_payload])

    print("Latency: {}ms".format(delay_total))

    # Confirmar el mensaje recibido
```

```
    ch.basic_ack(delivery_tag=method.delivery_tag)

# Conexión al broker

connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.0.185',
5672, '/', pika.PlainCredentials('admin', 'admin')))

channel = connection.channel()

# Crear la cola

channel.queue_declare(queue='ejemplo_qos2', durable=True)

# Configurar QoS

channel.basic_qos(prefetch_size=0, prefetch_count=1, all_channels=True)

# Suscribirse a la cola

channel.basic_consume(queue='ejemplo_qos2', on_message_callback=callback)

# Escuchar por mensajes entrantes

print(' [*] Esperando mensajes. Presione CTRL+C para salir')

channel.start_consuming()

# Cerrar la conexión

connection.close()
```

7.6. Anexo 6

Código empleado para la prueba de latencia para el protocolo MQTT para calidad de servicio QoS 0.

Equipo publicador

```
import paho.mqtt.client as mqtt

import time

# Inicializa el cliente MQTT

client = mqtt.Client()

# Conecta al broker MQTT

client.connect("192.168.0.184", 1883, 60)

# Envía 1000 paquetes de 64 B de tamaño

for i in range(1000):

    payload = "{}-{}".format(int(time.time() * 1000), "x" * (64 - len(str(int(time.time() * 1000)))))

    client.publish("test/topic", payload, qos=0)

    client.loop()

    time.sleep(5)

# Desconecta el cliente MQTT

client.disconnect()
```

Equipo suscriptor

```
import paho.mqtt.client as mqtt

import time

import csv

# Inicializa el cliente MQTT

client = mqtt.Client()
```

```
# Abrir un archivo CSV para registrar los tiempos de latencia y el tamaño de los paquetes
with open("latency_data.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Latency Total (ms)", "Packet Size (bytes)"])
    def on_message(client, userdata, msg):
        payload = msg.payload.decode("utf-8")
        payload_time = int(payload.split("-")[0])
        latency = time.time() * 1000 - payload_time
        print("Latency: {}ms".format(latency))
        writer.writerow([latency, len(payload)])
        f.flush()
    # Establece la función on_message como la función de devolución de llamada para cuando se reciba un mensaje
    client.on_message = on_message
    # Conecta al broker MQTT
    client.connect("192.168.0.184", 1883, 60)
    # Suscribirse al tópico
    client.subscribe("test/topic", qos=0)
    # Inicia el bucle de mensajes
    client.loop_forever()
```

7.7. Anexo 7

Código empleado para la prueba de latencia para el protocolo MQTT para calidad de servicio QoS 1.

Equipo publicador

```
import paho.mqtt.client as mqtt

import time

# Inicializa el cliente MQTT

client = mqtt.Client()

# Conecta al broker MQTT

client.connect("192.168.0.184", 1883, 60)

# Envía 1000 paquetes de 64 B de tamaño

for i in range(1000):

    payload = "{}-{}".format(int(time.time() * 1000), "x" * (64 - len(str(int(time.time() * 1000)))))

    client.publish("test/topic", payload, qos=1)

    client.loop()

    time.sleep(5)

# Desconecta el cliente MQTT

client.disconnect()
```

Equipo suscriptor

```
import paho.mqtt.client as mqtt

import time

import csv

# Inicializa el cliente MQTT

client = mqtt.Client()
```

```
# Abrir un archivo CSV para registrar los tiempos de latencia y el tamaño de los paquetes
with open("latency_data.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Latency Total (ms)", "Packet Size (bytes)"])
    def on_message(client, userdata, msg):
        payload = msg.payload.decode("utf-8")
        payload_time = int(payload.split("-")[0])
        latency = time.time() * 1000 - payload_time
        print("Latency: {}ms".format(latency))
        writer.writerow([latency, len(payload)])
        f.flush()
    # Establece la función on_message como la función de devolución de llamada para cuando se reciba un mensaje
    client.on_message = on_message
    # Conecta al broker MQTT
    client.connect("192.168.0.184", 1883, 60)
    # Suscribirse al tópico
    client.subscribe("test/topic", qos=1)
    # Inicia el bucle de mensajes
    client.loop_forever()
```

7.8. Anexo 8

Código empleado para la prueba de latencia MQTT para calidad de servicio QoS 2

Equipo publicador

```
import paho.mqtt.client as mqtt

import time

# Inicializa el cliente MQTT

client = mqtt.Client()

# Conecta al broker MQTT

client.connect("192.168.0.184", 1883, 60)

# Envía 1000 paquetes de 64 B de tamaño

for i in range(1000):

    payload = "{}-{}".format(int(time.time() * 1000), "x" * (64 - len(str(int(time.time() * 1000)))))

    client.publish("test/topic", payload, qos=2)

    client.loop()

    #time.sleep(5)

# Desconecta el cliente MQTT

client.disconnect()
```

Equipo suscriptor

```
import paho.mqtt.client as mqtt

import time

import csv

# Inicializa el cliente MQTT

client = mqtt.Client()
```

```
# Abrir un archivo CSV para registrar los tiempos de latencia y el tamaño de los paquetes
with open("latency_data.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Latency Total (ms)", "Packet Size (bytes)"])
    def on_message(client, userdata, msg):
        payload = msg.payload.decode("utf-8")
        payload_time = int(payload.split("-")[0])
        latency = time.time() * 1000 - payload_time
        print("Latency: {}ms".format(latency))
        writer.writerow([latency, len(payload)])
        f.flush()
    # Establece la función on_message como la función de devolución de llamada para cuando se reciba un mensaje
    client.on_message = on_message
    # Conecta al broker MQTT
    client.connect("192.168.0.184", 1883, 60)
    # Suscribirse al tópico
    client.subscribe("test/topic", qos=2)
    # Inicia el bucle de mensajes
    client.loop_forever()
```

7.9. Anexo 9

Código para diagrama de cajas y bigote para comparar los diferentes niveles de calidad.

```
import matplotlib.pyplot as plt

import pandas as pd

# Cargar los datos desde los archivos Excel

df_qos0 = pd.read_excel('amqp0.xlsx')

df_qos1 = pd.read_excel('amqp1.xlsx')

df_qos2 = pd.read_excel('amqp2.xlsx')

# Crear una lista de listas con los valores de delay para cada tamaño de paquete y QoS

data_qos0 = [df_qos0['64B'].tolist(),

        df_qos0['256B'].tolist(),

        df_qos0['1kB'].tolist(),

        df_qos0['4kB'].tolist(),

        df_qos0['16kB'].tolist(),

        df_qos0['64kB'].tolist()]

data_qos1 = [df_qos1['64B'].tolist(),

        df_qos1['256B'].tolist(),

        df_qos1['1kB'].tolist(),

        df_qos1['4kB'].tolist(),

        df_qos1['16kB'].tolist(),

        df_qos1['64kB'].tolist()]

data_qos2 = [df_qos2['64B'].tolist(),

        df_qos2['256B'].tolist(),

        df_qos2['1kB'].tolist(),

        df_qos2['4kB'].tolist(),
```

```
        df_qos2['16kB'].tolist(),
        df_qos2['64kB'].tolist()]
# Crear el boxplot con los datos
fig, ax = plt.subplots(figsize=(8, 6), dpi=120)
# Graficar los datos de QoS 0 en color rojo
ax.boxplot(data_qos0, positions=[1,2,3,4,5,6], widths=0.2, patch_artist=True,
boxprops=dict(facecolor='red'))
ax.text(0.25, 0.9, 'QoS 0', ha='center', transform=ax.transAxes, color='red')
# Graficar los datos de QoS 1 en color verde
ax.boxplot(data_qos1, positions=[1.3,2.3,3.3,4.3,5.3,6.3], widths=0.2, patch_artist=True,
boxprops=dict(facecolor='green'))
ax.text(0.5, 0.9, 'QoS 1', ha='center', transform=ax.transAxes, color='green')
# Graficar los datos de QoS 2 en color azul
ax.boxplot(data_qos2, positions=[1.6,2.6,3.6,4.6,5.6,6.6], widths=0.2, patch_artist=True,
boxprops=dict(facecolor='blue'))
ax.text(0.75, 0.9, 'QoS 2', ha='center', transform=ax.transAxes, color='blue')
# Añadir etiquetas al eje x
ax.set_xticks([1.3,2.3,3.3,4.3,5.3,6.3])
ax.set_xticklabels(['64B', '256B', '1kB', '4kB', '16kB', '64kB'])
# Añadir título y etiquetas a los ejes
ax.set_title('Rendimiento (throughput) - AMQP QoS')
ax.set_xlabel('Tamaño de payload')
ax.set_ylabel('Rendimiento (MB/s)')
# Ajustar el espacio en la figura
fig.tight_layout()
```

```
# Mostrar el gráfico
plt.show()
```

7.10. Anexo 10

Código para diagrama de cajas y bigote para un solo nivel de calidad

```
import matplotlib.pyplot as plt
import pandas as pd
# Cargar los datos desde el archivo Excel
df = pd.read_excel('mqtt2.xlsx')
# Crear una lista de listas con los valores de delay para cada tamaño de paquete
data = [
    df['64B'].tolist(),
    df['256B'].tolist(),
    df['1kB'].tolist(),
    df['4kB'].tolist(),
    df['16kB'].tolist(),
    df['64kB'].tolist()
]
# Crear el boxplot con los datos
fig, ax = plt.subplots()
ax.boxplot(data)
# Añadir etiquetas al eje x
ax.set_xticklabels(['64B', '256B', '1kB', '4kB', '16kB', '64kB'])
# Añadir título y etiquetas a los ejes
ax.set_title('Rendimiento (throughput) - MQTT QoS 2')
ax.set_xlabel('Tamaño de payload')
ax.set_ylabel('Rendimiento (mbps)')
# Mostrar el gráfico
```

```
plt.show()
```

7.11. Anexo 11

Código para determinar estadísticos descriptivos

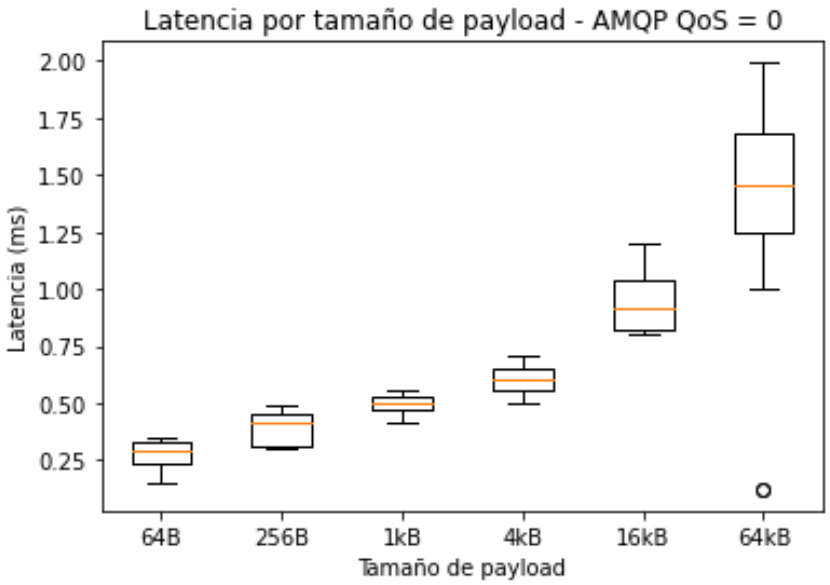
```
import pandas as pd
import openpyxl
# Cargar el archivo de Excel
wb = openpyxl.load_workbook('amqp2.xlsx')
# Seleccionar la hoja de Excel que contiene los datos
ws = wb['Sheet1']
# Crear un diccionario para almacenar los resultados
resultados = {}
# Iterar sobre cada columna de datos
for col in ws.iter_cols(min_col=22, max_col=27):
    # Obtener el encabezado de la columna
    header = col[0].value
    # Convertir los valores de la columna en una lista
    values = [cell.value for cell in col[1:]]
    # Calcular los estadísticos descriptivos de la columna
    media = round(pd.Series(values).mean(), 2)
    mediana = round(pd.Series(values).median(), 2)
    desviacion_estandar = round(pd.Series(values).std(), 2)
    minimo = round(pd.Series(values).min(), 2)
    maximo = round(pd.Series(values).max(), 2)
    # Agregar los resultados al diccionario
    resultados[header] = [media, mediana, desviacion_estandar, minimo, maximo]
# Crear un DataFrame de pandas a partir del diccionario de resultados
```

```
df = pd.DataFrame.from_dict(resultados, orient='index',
                            columns=['Media', 'Mediana', 'Desv. Estándar', 'Mínimo', 'Máximo'])
# Ordenar el DataFrame por el tamaño de payload (de menor a mayor)
df = df.reindex(['64B', '256B', '1kB', '4kB', '16kB', '64kB'])
# Mostrar el DataFrame en formato de tabla
print(df.to_string())
```

7.12. Anexo 12

Las figuras 46, 47 y 48 muestran los resultados obtenidos de la prueba de latencia para el protocolo AMQP**,** mientras que en las tablas 7, 8 y 9 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 46. Latencia por tamaño de *payload* – AMQP QoS 0



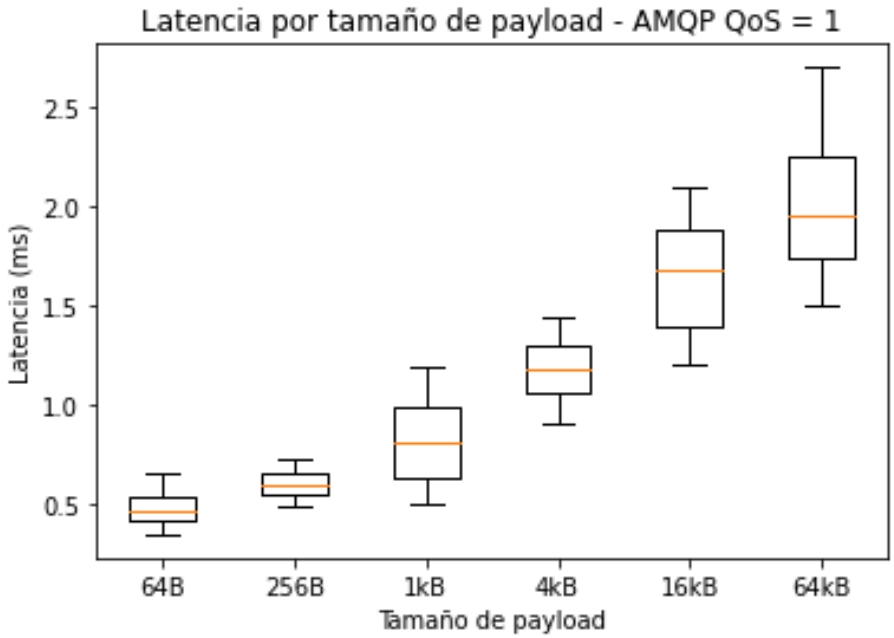
Fuente: Elaboración propia.

Tabla 7. Estadísticos descriptivos de latencia para el protocolo AMQP QoS 0

Tamaño del payload	Media (ms)	Mediana (ms)	Desviación estándar (ms)	Mínimo (ms)	Máximo (ms)
64B	0,28	0,30	0,06	0,15	0,35
256B	0,38	0,41	0,07	0,30	0,49
1kB	0,50	0,50	0,04	0,41	0,56
4kB	0,60	0,60	0,06	0,50	0,71
16kB	0,94	0,92	0,12	0,80	1,20
64kB	1,46	1,45	0,28	0,12	1,99

Fuente: Elaboración propia.

Figura 47. Latencia por tamaño de *payload*– AMQP QoS 1



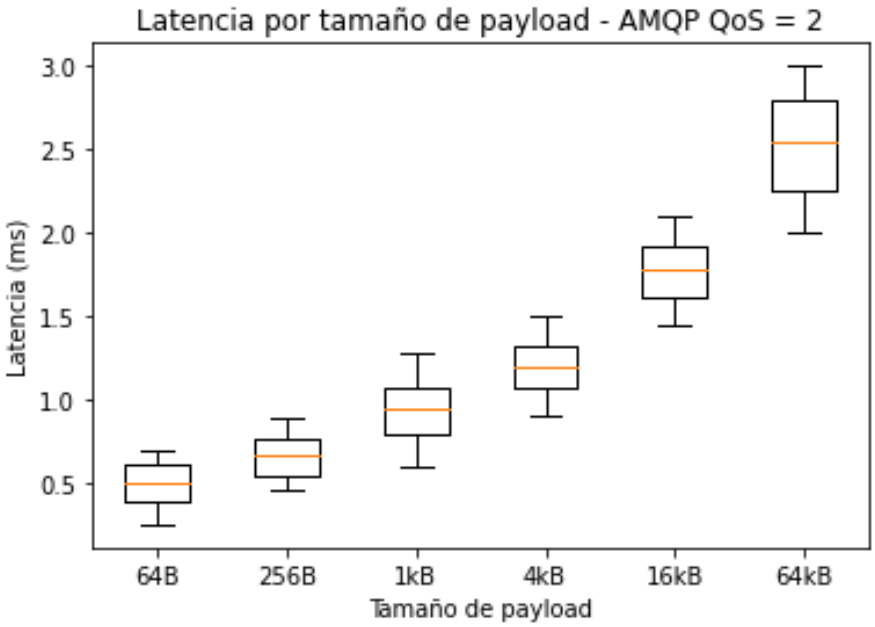
Fuente: Elaboración propia.

Tabla 8. Estadísticos descriptivos de latencia para el protocolo AMQP QoS 1

Tamaño del payload	**Media (ms)**	**Mediana (ms)**	**Desviación estándar (ms)**	**Mínimo (ms)**	**Máximo (ms)**
64B	0,48	0,47	0,08	0,35	0,66
256B	0,60	0,60	0,07	0,50	0,73
1kB	0,82	0,81	0,21	0,50	1,20
4kB	1,18	1,19	0,15	0,90	1,44
16kB	1,66	1,68	0,27	1,21	2,10
64kB	2,01	1,96	0,33	1,50	2,71

Fuente: Elaboración propia.

Figura 48. Latencia por tamaño de *payload* – AMQP QoS 2



Fuente: Elaboración propia.

Tabla 9. Estadísticos descriptivos de latencia para el protocolo AMQP QoS 2

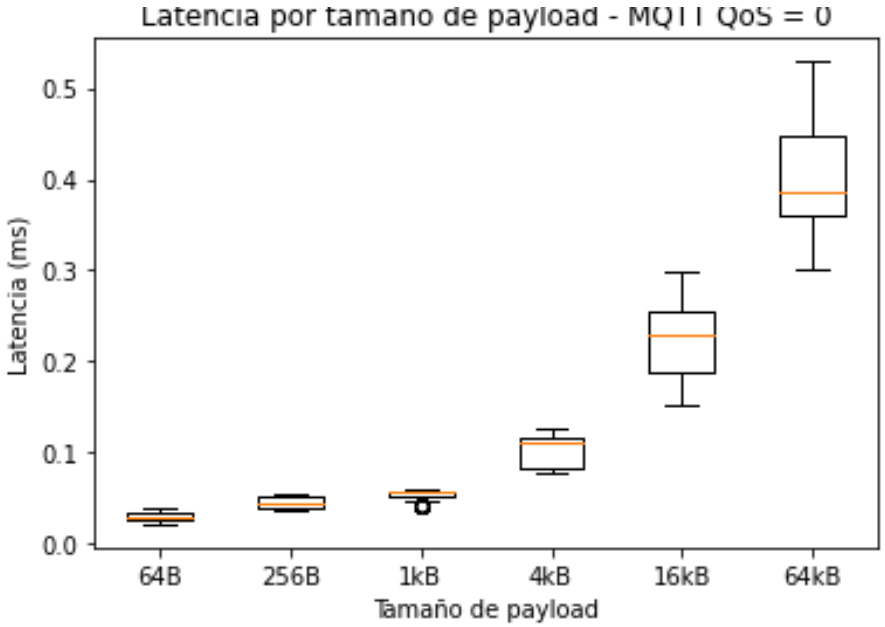
Tamaño del payload	**Media (ms)**	**Mediana (ms)**	**Desviación estándar (ms)**	**Mínimo (ms)**	**Máximo (ms)**
64B	0,50	0,51	0,13	0,25	0,70
256B	0,66	0,67	0,14	0,45	0,90
1kB	0,93	0,95	0,18	0,60	1,27
4kB	1,20	1,20	0,16	0,90	1,51
16kB	1,77	1,78	0,18	1,44	2,10
64kB	2,52	2,54	0,30	2,00	3,00

Fuente: Elaboración propia.

7.13. Anexo 13

En las figuras 49, 50 y 51 se muestran los resultados obtenidos de la prueba de latencia para el protocolo MQTT, mientras que en las tablas 10, 11 y 12 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 49. Latencia por tamaño de *payload* – MQTT QoS 0



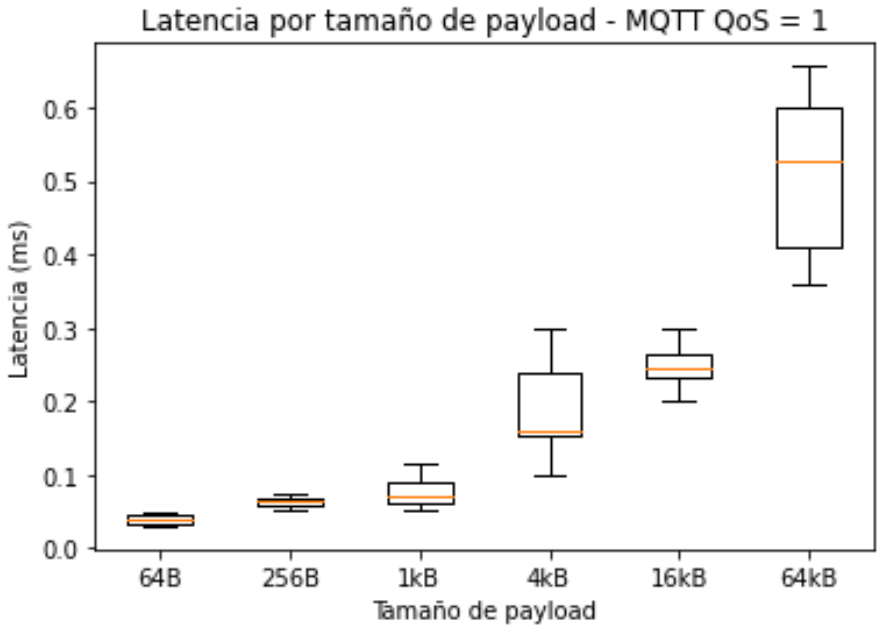
Fuente: Elaboración propia.

Tabla 10. Estadísticos descriptivos de latencia para el protocolo MQTT QoS 0

Tamaño del payload	**Media (ms)**	**Mediana (ms)**	**Desviación estándar (ms)**	**Mínimo (ms)**	**Máximo (ms)**
64B	0,03	0,03	0,00	0,02	0,04
256B	0,05	0,05	0,01	0,04	0,05
1kB	0,05	0,06	0,00	0,04	0,06
4kB	0,10	0,11	0,02	0,08	0,13
16kB	0,22	0,23	0,04	0,15	0,30
64kB	0,40	0,39	0,06	0,30	0,53

Fuente: Elaboración propia.

Figura 50. Latencia por tamaño de *payload* – MQTT QoS 1



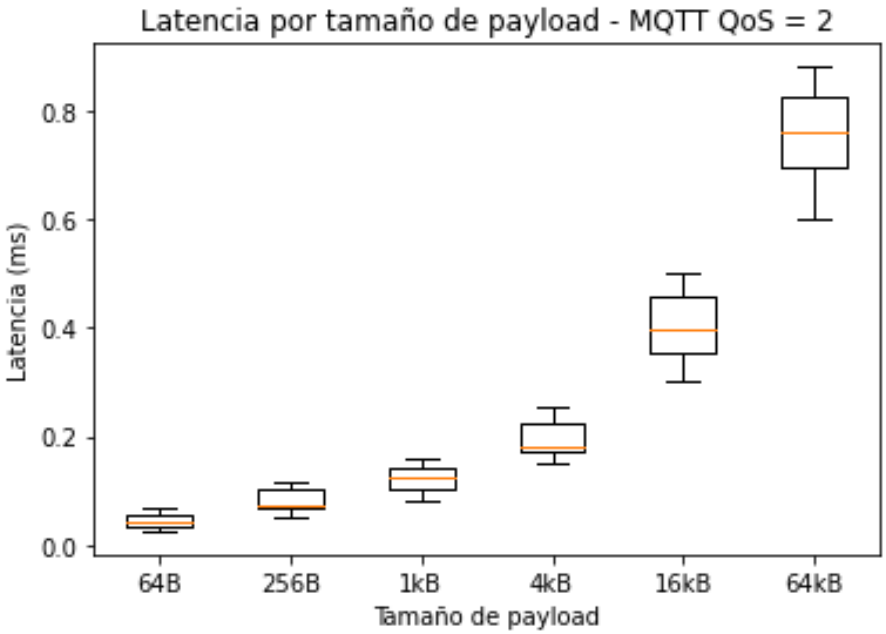
Fuente: Elaboración propia.

Tabla 11. Estadísticos descriptivos de latencia para el protocolo MQTT QoS 1

Tamaño del payload	**Media (ms)**	**Mediana (ms)**	**Desviación estándar (ms)**	**Mínimo (ms)**	**Máximo (ms)**
64B	0,04	0,04	0,01	0,03	0,05
256B	0,06	0,07	0,01	0,05	0,07
1kB	0,08	0,07	0,02	0,05	0,12
4kB	0,18	0,16	0,05	0,10	0,30
16kB	0,25	0,24	0,03	0,20	0,30
64kB	0,51	0,53	0,10	0,36	0,66

Fuente: Elaboración propia.

Figura 51. Latencia por tamaño de *payload* – MQTT QoS 2



Fuente: Elaboración propia.

Tabla 12. Estadísticos descriptivos de latencia para el protocolo MQTT QoS 2

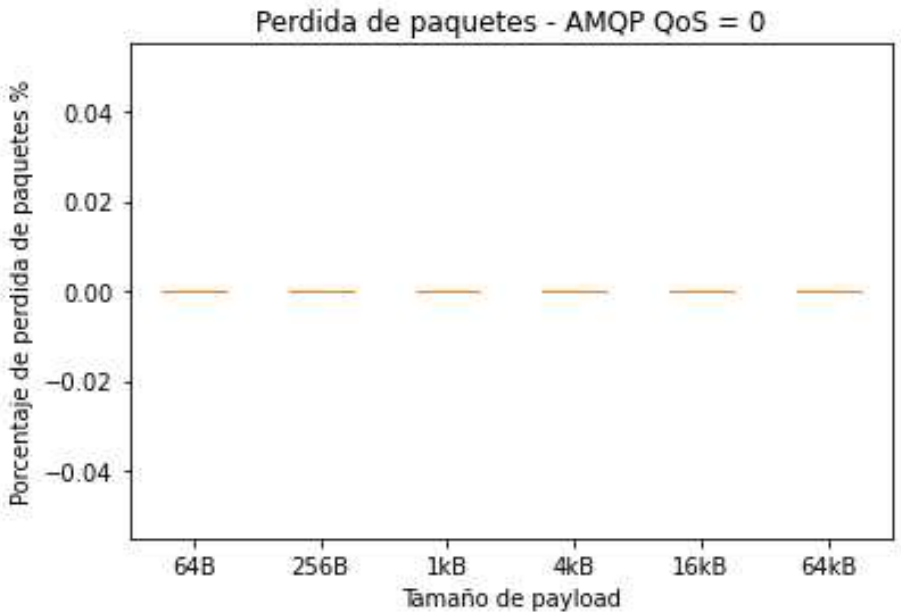
Tamaño del payload	**Media (ms)**	**Mediana (ms)**	**Desviación estándar (ms)**	**Mínimo (ms)**	**Máximo (ms)**
64B	0,05	0,04	0,01	0,03	0,07
256B	0,08	0,07	0,02	0,05	0,11
1kB	0,12	0,12	0,02	0,08	0,16
4kB	0,19	0,18	0,03	0,15	0,25
16kB	0,40	0,40	0,06	0,30	0,50
64kB	0,76	0,76	0,08	0,60	0,88

Fuente: Elaboración propia.

7.14. Anexo 14

En las figuras 52, 53 y 54 se muestran los resultados obtenidos de la prueba de perdida de paquetes para el protocolo AMQP, mientras que en las tablas 13, 14 y 15 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 52. Perdida de paquetes para el protocolo AMQP QoS 0



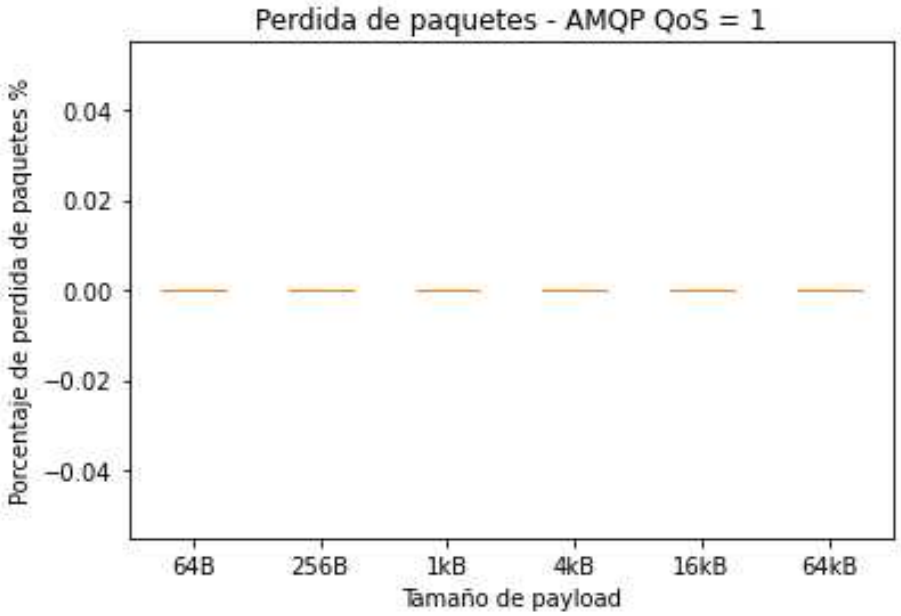
Fuente: Elaboración propia.

Tabla 13. Estadísticos descriptivos de perdida de paquetes protocolo AMQP QoS 0

Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0	0	0	0	0
256B	0	0	0	0	0
1kB	0	0	0	0	0
4kB	0	0	0	0	0
16kB	0	0	0	0	0
64kB	0	0	0	0	0

Fuente: Elaboración propia.

Figura 53. Perdida de paquetes para el protocolo AMQP QoS 1



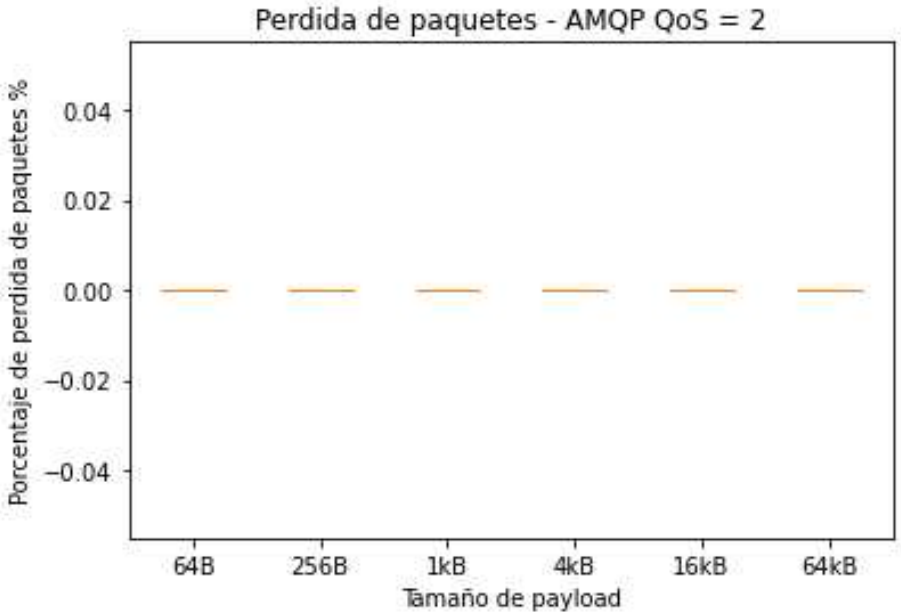
Fuente: Elaboración propia.

Tabla 14. Estadísticos descriptivos de perdida de paquetes protocolo AMQP QoS 1

Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0	0	0	0	0
256B	0	0	0	0	0
1kB	0	0	0	0	0
4kB	0	0	0	0	0
16kB	0	0	0	0	0
64kB	0	0	0	0	0

Fuente: Elaboración propia.

Figura 54. Perdidas de paquetes para el protocolo AMQP QoS 2



Fuente: Elaboración propia.

Tabla 15. Estadísticos descriptivos de perdida de paquetes protocolo AMQP QoS 2

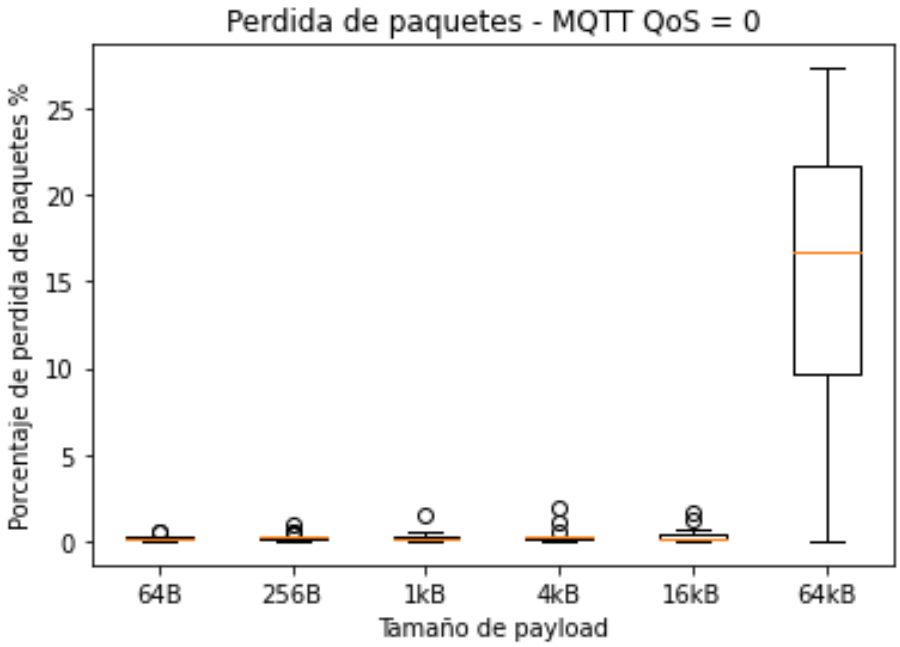
Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0	0	0	0	0
256B	0	0	0	0	0
1kB	0	0	0	0	0
4kB	0	0	0	0	0
16kB	0	0	0	0	0
64kB	0	0	0	0	0

Fuente: Elaboración propia.

7.15. Anexo 15

En las figuras 55, 56 y 57 muestran los resultados obtenidos de la prueba de perdida de paquetes para el protocolo MQTT, mientras que en las tablas 16, 17 y 18 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 55. Perdidas de paquetes para el protocolo MQTT QoS 0



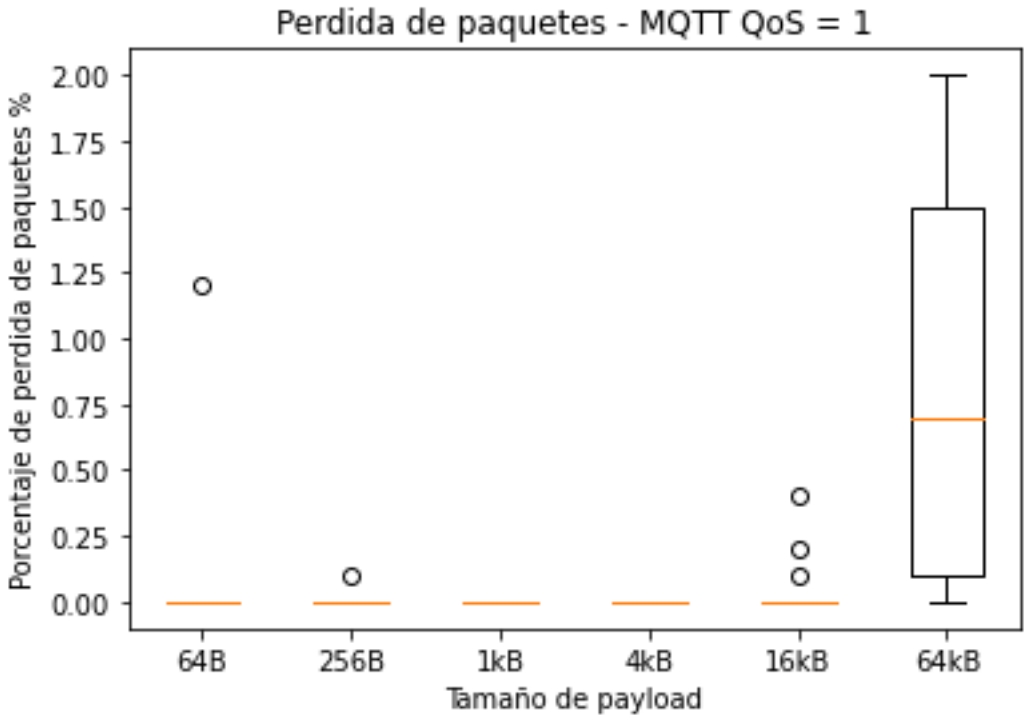
Fuente: Elaboración propia.

Tabla 16. Estadísticos descriptivos de perdida de paquetes protocolo MQTT QoS 0

Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0.17	0.1	0.13	0	0.5
256B	0.22	0.2	0.22	0	1.0
1kB	0.24	0.1	0.31	0	1.5
4kB	0.27	0.2	0.40	0	1.9
16kB	0.30	0.1	0.39	0	1.6
64kB	15.72	16.7	7.67	0	27.3

Fuente: Elaboración propia.

Figura 56. Perdidas de paquetes para el protocolo MQTT QoS 1



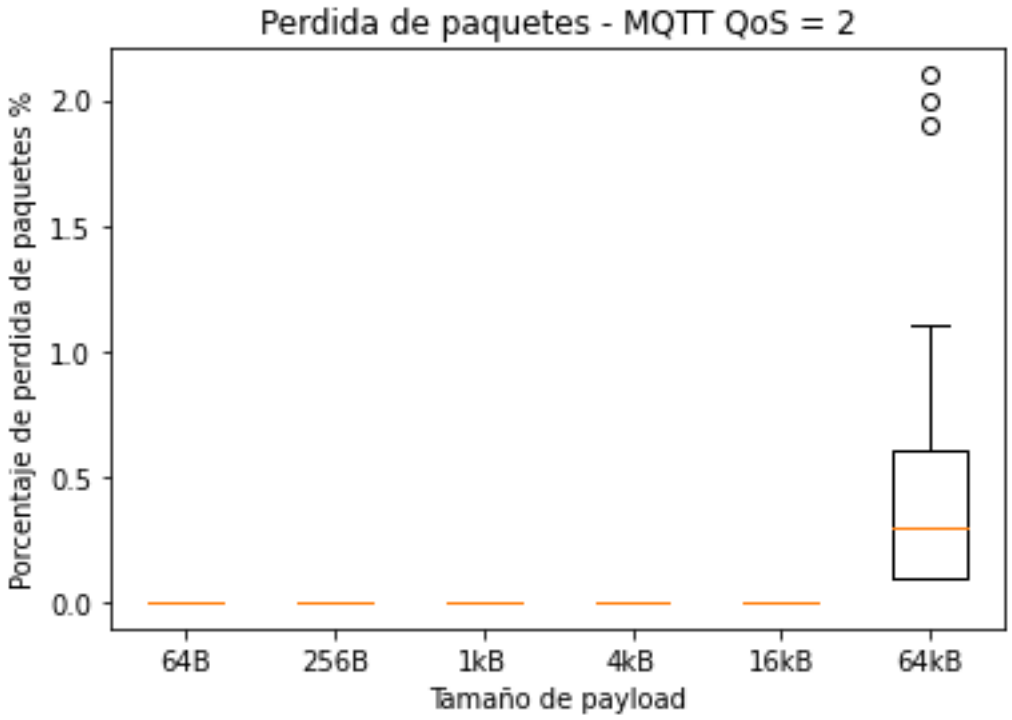
Fuente: Elaboración propia.

Tabla 17. Estadísticos descriptivos de perdida de paquetes protocolo MQTT QoS 1

Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0.05	0	0.24	0	1.2
256B	0	0	0.02	0	0.1
1kB	0	0	0	0	0
4kB	0	0	0	0	0
16kB	0.03	0	0.09	0	0.4
64kB	0.84	0.7	0.71	0	2.0

Fuente: Elaboración propia

Figura 57. Perdidas de paquetes para el protocolo MQTT QoS 2



Fuente: Elaboración propia.

Tabla 18. Estadísticos descriptivos de perdida de paquetes protocolo MQTT QoS 2

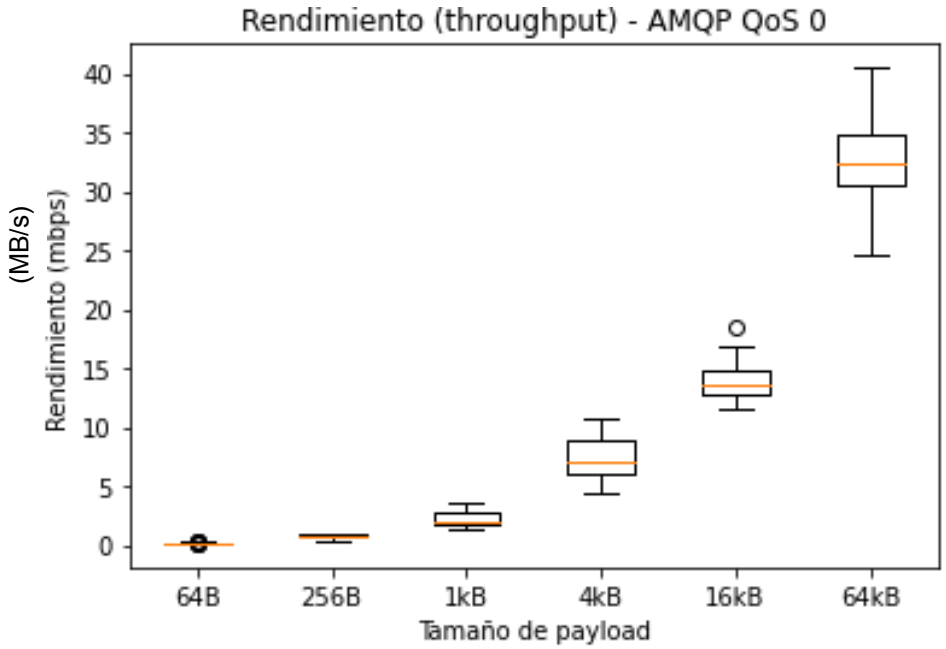
Tamaño del payload	**Media (%)**	**Mediana (%)**	**Desviación estándar (%)**	**Mínimo (%)**	**Máximo (%)**
64B	0	0	0	0	0
256B	0	0	0	0	0
1kB	0	0	0	0	0
4kB	0	0	0	0	0
16kB	0	0	0	0	0
64kB	0.54	0.3	0.62	0.1	2.10

Fuente: Elaboración propia.

7.16. Anexo 16

Las figuras 58, 59 y 60 muestran los resultados obtenidos de la prueba de rendimiento para el protocolo AMQP, mientras que en las tablas 19, 20 y 21 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 58. Rendimiento (*throughput*) para el protocolo AMQP QoS 0



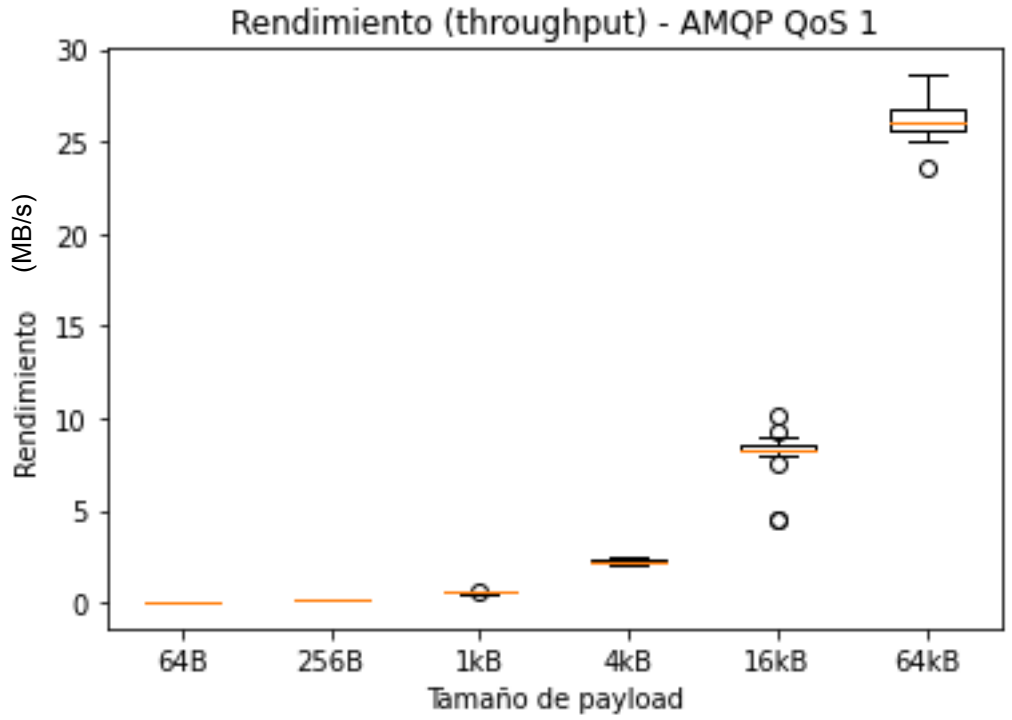
Fuente: Elaboración propia.

Tabla 19. Estadísticos descriptivos de rendimiento para el protocolo AMQP QoS 0

Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.20	0.20	0.04	0.13	0.32
256B	0.74	0.74	0.18	0.39	1.01
1kB	2.20	1.85	0.68	1.39	3.56
4kB	7.49	7.03	1.79	4.46	10.79
16kB	14.05	13.57	1.75	11.63	18.52
64kB	32.50	32.27	3.37	24.51	40.46

Fuente: Elaboración propia.

Figura 59. Rendimiento (*throughput*) para el protocolo AMQP QoS 1



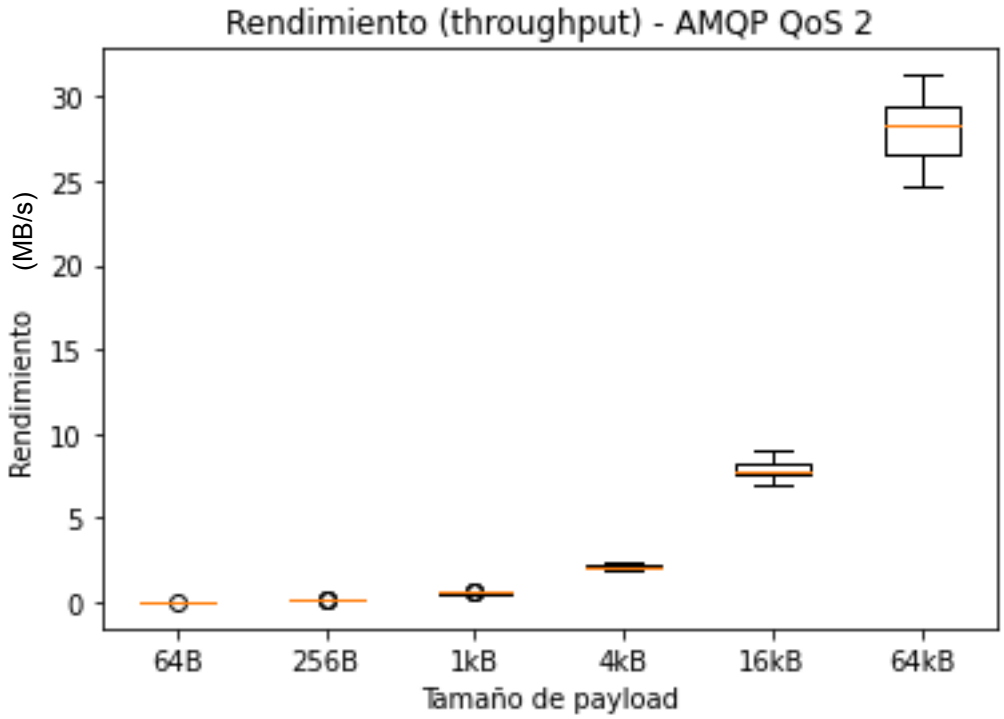
Fuente: Elaboración propia.

Tabla 20. Estadísticos descriptivos de rendimiento para el protocolo AMQP QoS 1

Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.03	0.03	0.01	0.02	0.04
256B	0.11	0.11	0.01	0.10	0.14
1kB	0.55	0.56	0.05	0.48	0.64
4kB	2.24	2.24	0.11	1.98	2.41
16kB	8.13	8.30	1.20	4.43	10.14
64Kb	26.20	25.98	1.08	23.53	28.61

Fuente: Elaboración propia.

Figura 60. Rendimiento (*throughput*) para el protocolo AMQP QoS 2



Fuente: Elaboración propia.

Tabla 21. Estadísticos descriptivos de rendimiento para el protocolo AMQP QoS 2

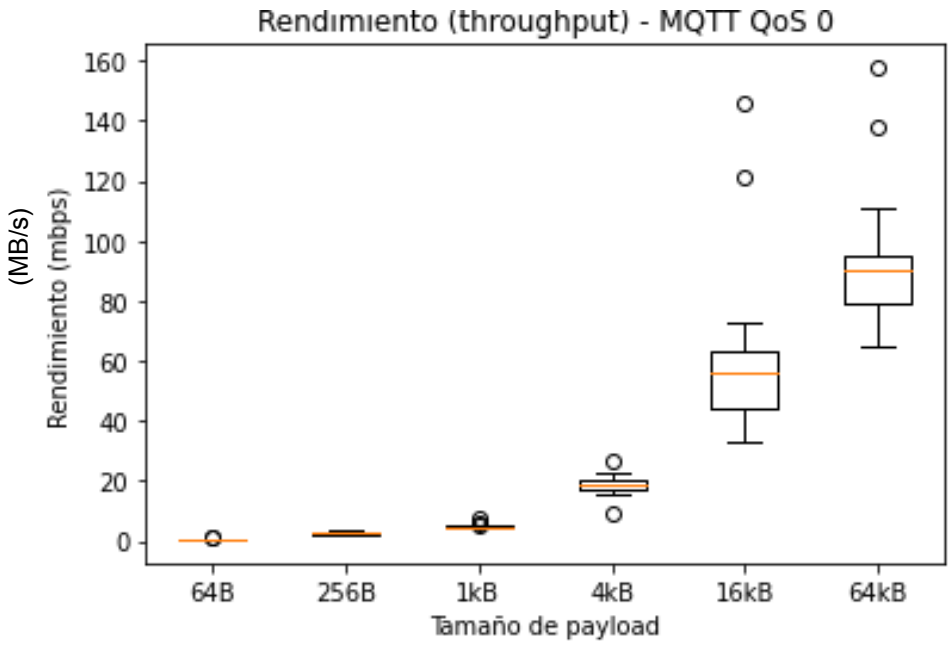
Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.04	0.04	0.00	0.03	0.05
256B	0.16	0.16	0.02	0.14	0.21
1kB	0.59	0.57	0.05	0.54	0.73
4kB	2.10	2.09	0.12	1.91	2.33
16kB	7.88	7.81	0.51	6.90	9.06
64kB	28.09	28.24	1.75	24.64	31.27

Fuente: Elaboración propia.

7.17. Anexo 17

En las figuras 61, 62 y 63 se muestran los resultados obtenidos de la prueba de rendimiento para el protocolo MQTT, mientras que en las tablas 22, 23 y 24 se muestran los estadísticos descriptivos que se obtuvieron.

Figura 61. Rendimiento (*throughput*) para el protocolo MQTT QoS 0



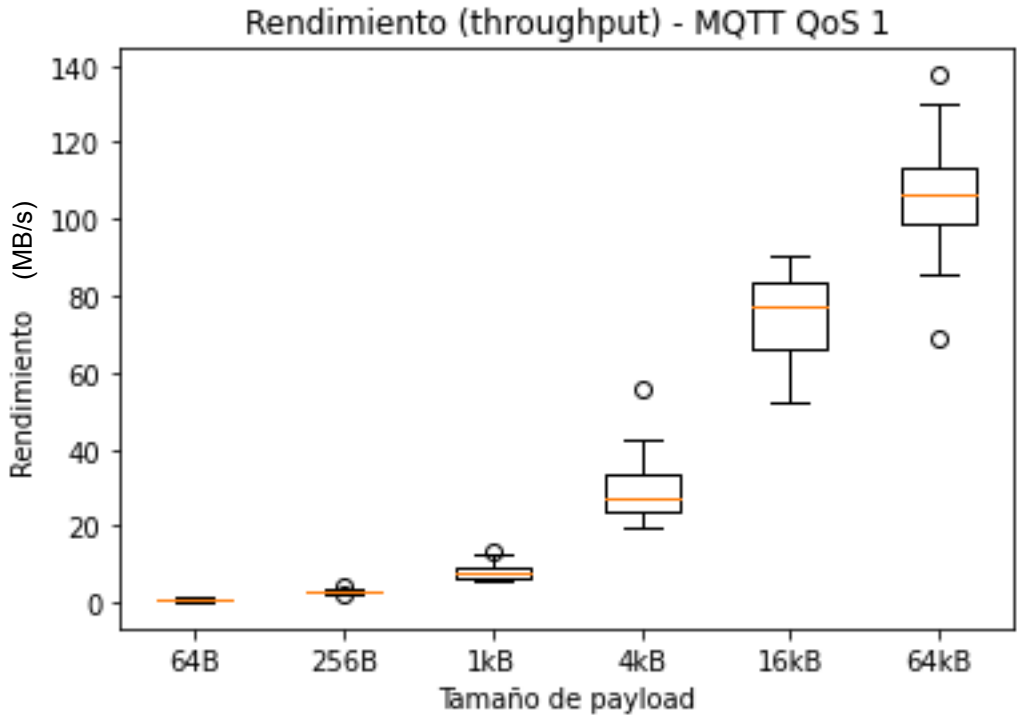
Fuente: Elaboración propia

Tabla 22. Estadísticos descriptivos de rendimiento para el protocolo MQTT QoS 0

Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.66	0.64	0.16	0.47	1.27
256B	2.49	2.51	0.32	1.95	3.21
1kB	4.78	4.52	0.78	4.04	7.87
4kB	18.76	18.53	3.32	9.28	26.90
16kB	60.41	56.42	24.57	33.24	145.61
64kB	90.98	89.98	20.30	64.89	157.65

Fuente: Elaboración propia.

Figura 62. Rendimiento (*throughput*) para el protocolo MQTT QoS 1



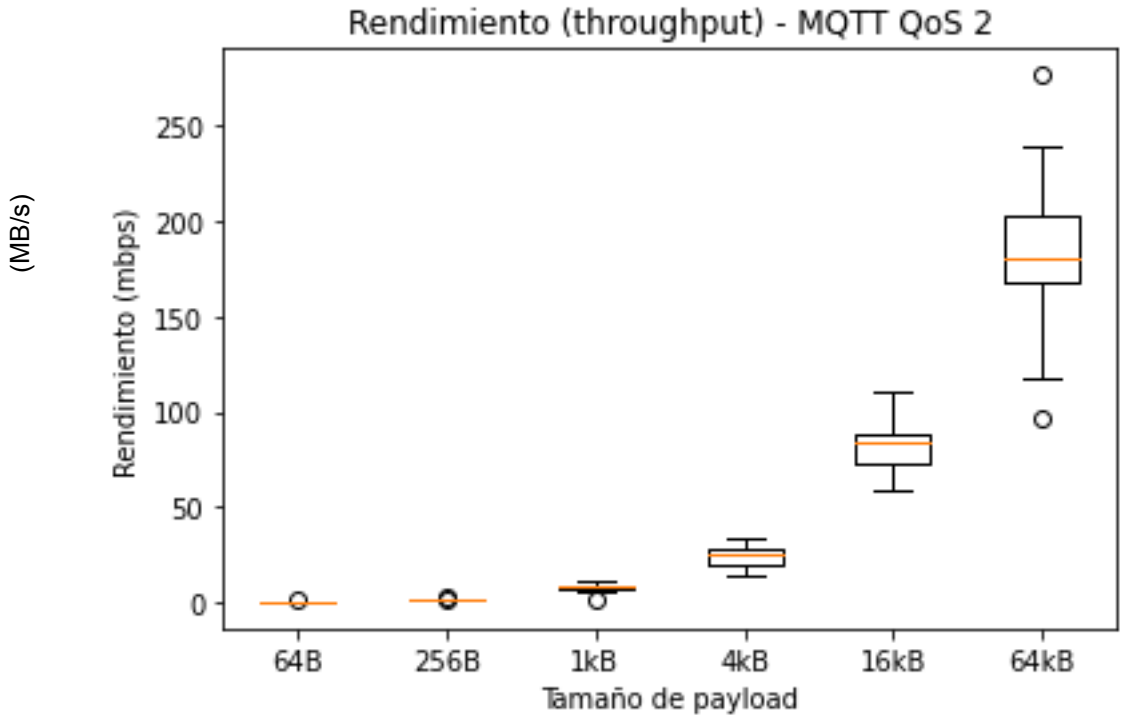
Fuente: Elaboración propia.

Tabla 23. Estadísticos descriptivos del rendimiento para el protocolo MQTT QoS 1

Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.86	0.70	0.40	0.25	1.80
256B	2.96	2.96	0.46	1.95	4.27
1kB	8.33	7.95	2.22	5.69	13.67
4kB	29.51	27.23	8.04	19.54	55.71
16kB	74.99	77.24	11.22	52.13	90.21
64kB	106.12	106.46	14.27	69.20	137.52

Fuente: Elaboración propia.

Figura 63. Rendimiento (*throughput*) para el protocolo MQTT QoS 2



Fuente: Elaboración propia.

Tabla 24. Estadísticos descriptivos del rendimiento para el protocolo MQTT QoS 2

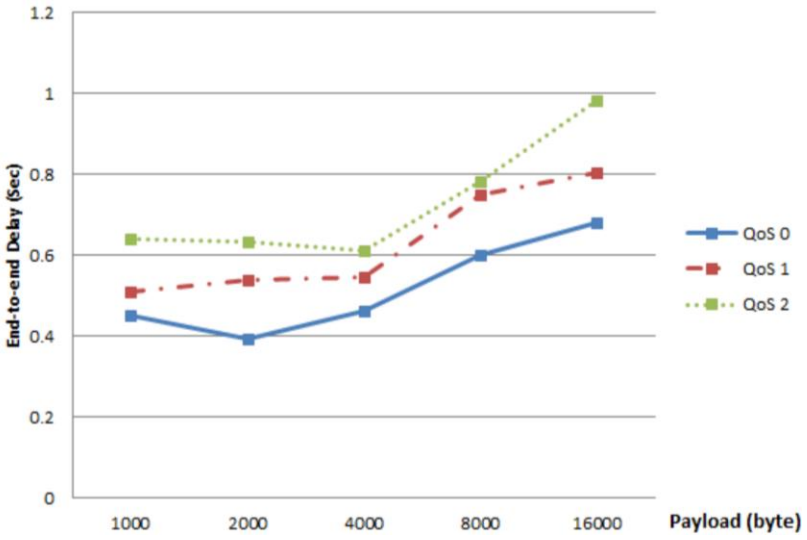
Tamaño del payload	**Media (mbps)**	**Mediana (mbps)**	**Desviación estándar (mbps)**	**Mínimo (mbps)**	**Máximo (mbps)**
64B	0.50	0.47	0.16	0.27	0.94
256B	1.57	1.51	0.31	1.19	2.58
1kB	7.73	7.88	1.89	0.91	10.83
4kB	25.20	25.54	5.87	14.77	34.01
16kB	82.49	83.63	12.88	59.39	111.13
64kB	181.17	180.33	39.94	96.06	276.32

Fuente: Elaboración propia.

7.18. Anexo 18

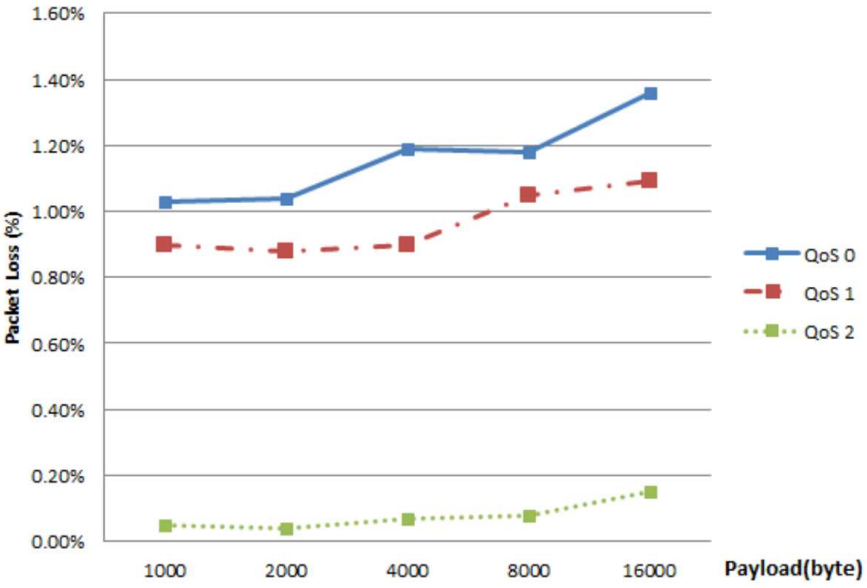
Las figuras 64 y 65 se muestran los resultados obtenidos de los experimentos desarrollados por en el artículo “Correlation Analysis of MQTT Loss and Delay According to QoS” (Lee, Kim, Hong, & Ju, 2013).

Figura 64. Resultado del análisis de latencia de la red inalámbrica



Fuente: Tomado de Lee, Kim, Hong, & Ju (2013).

Figura 65. Resultado del análisis de pérdida de mensajes de red Inalámbrica

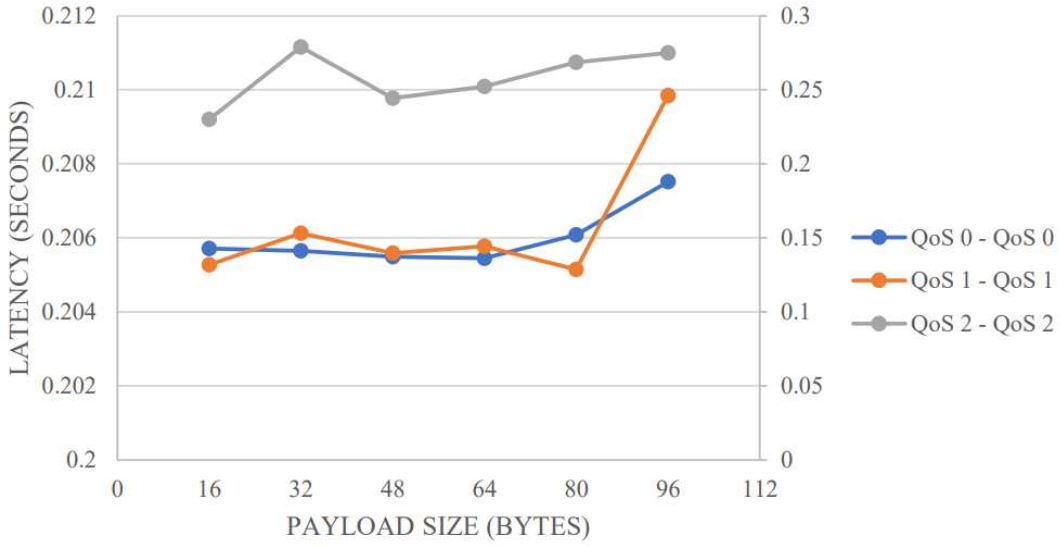


Fuente: Tomado de Lee, Kim, Hong, & Ju (2013).

7.19. Anexo 19

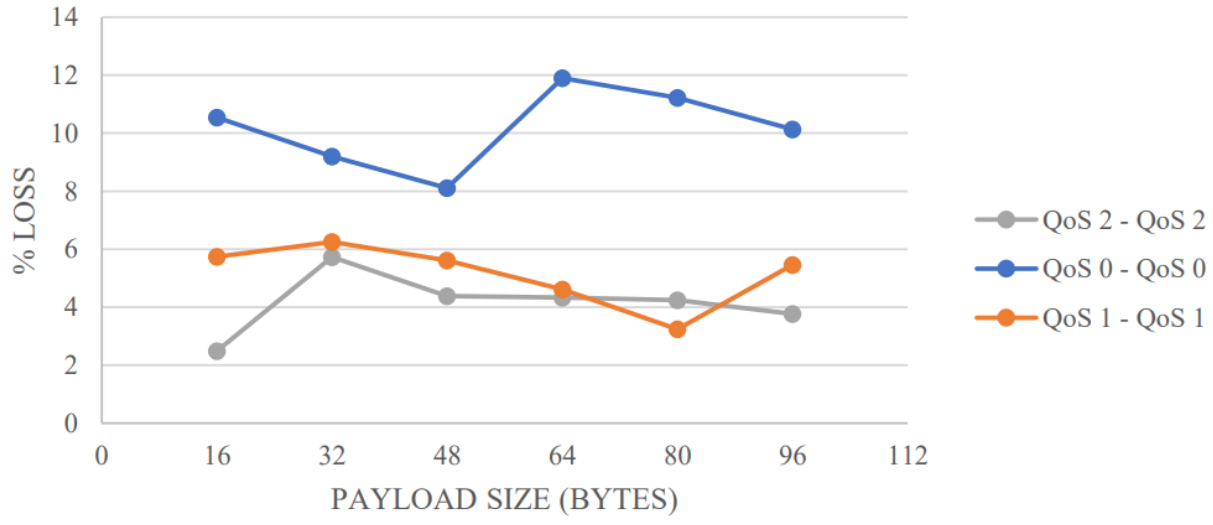
Las figuras 66, 67, 68, 69, 70, 71 y 72 muestran los resultados obtenidos de los experimentos desarrollados en la tesis: A real time demonstrative analysis of lightweight payload encryption in resource constrained devices based on MQTT (Menyah, 2017).

Figura 66. Resultado del análisis de latencia de extremo a extremo promedio para carga útil no encriptada



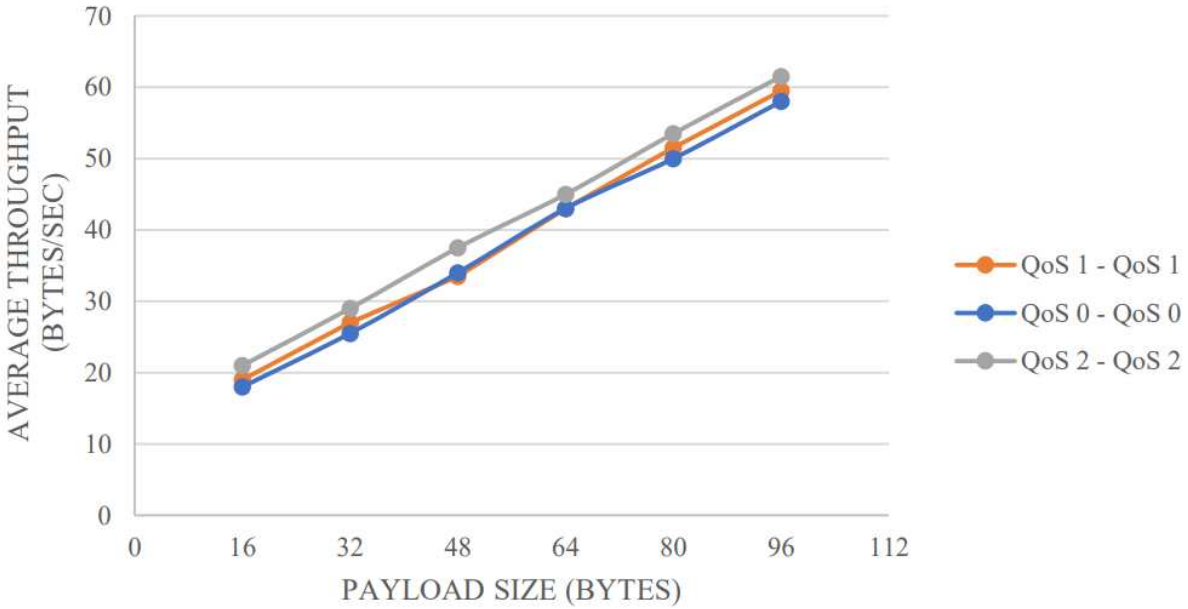
Fuente: Tomado de Nanabayin Menyah (2017).

Figura 67. Resultado del análisis del porcentaje de pérdida de carga útil no cifrada.



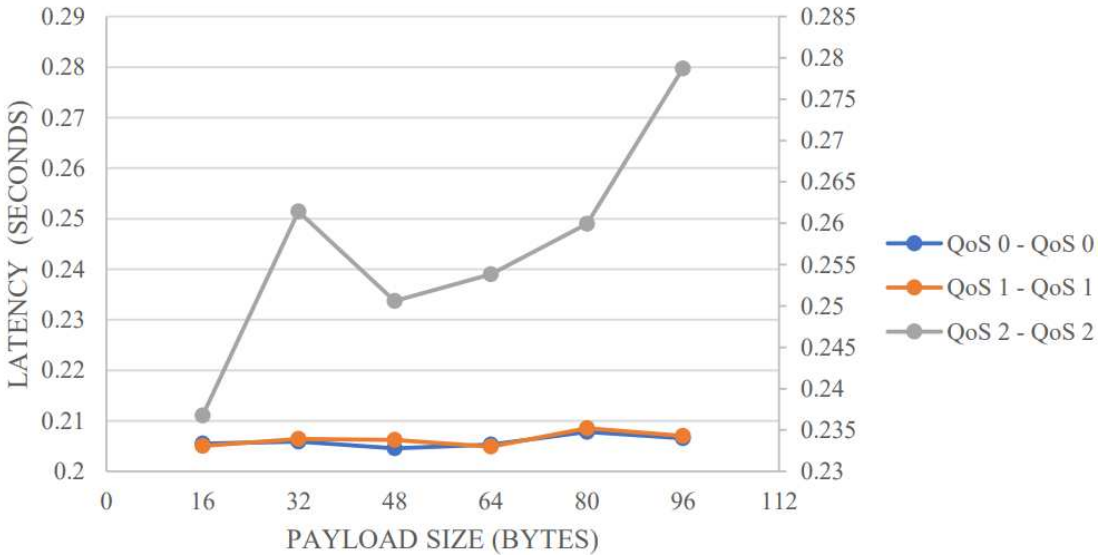
Fuente: Tomado de Nanabayin Menyah (2017).

Figura 68. Resultado del análisis del rendimiento de la carga útil no cifrada.



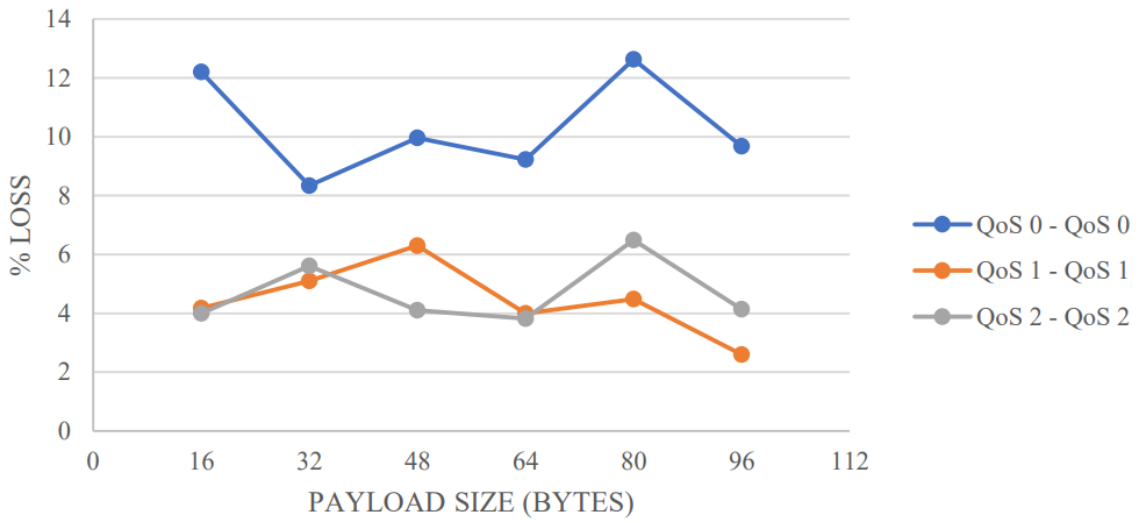
Fuente: Tomado de Nanabayin Menyah (2017).

Figura 69. Resultado del análisis de latencia de extremo a extremo promedio de la carga útil cifrada.



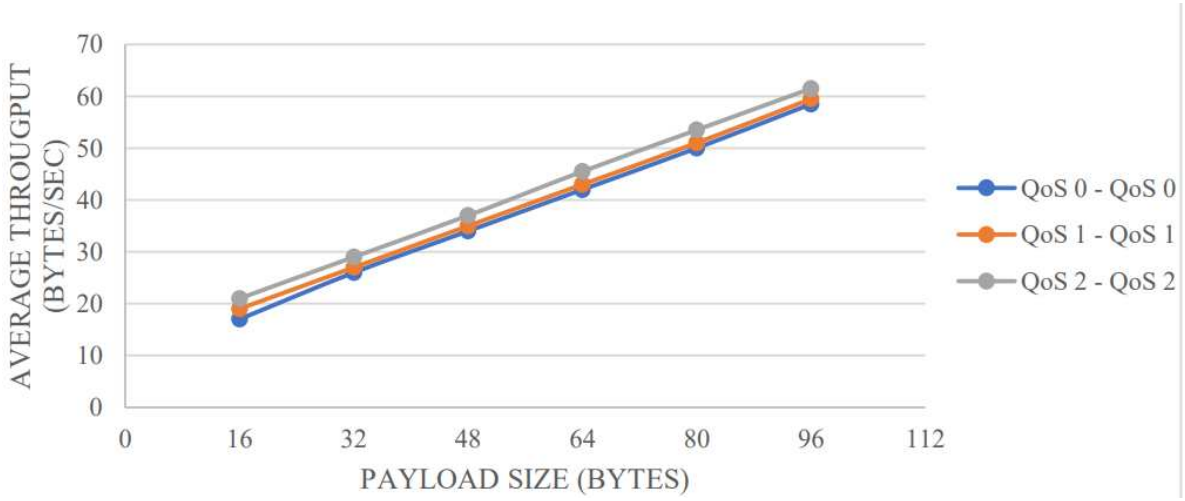
Fuente: Tomado de Nanabayin Menyah (2017).

Figura 70. Resultado del análisis del porcentaje de pérdida de la carga útil cifrada con AES-128 BITS.



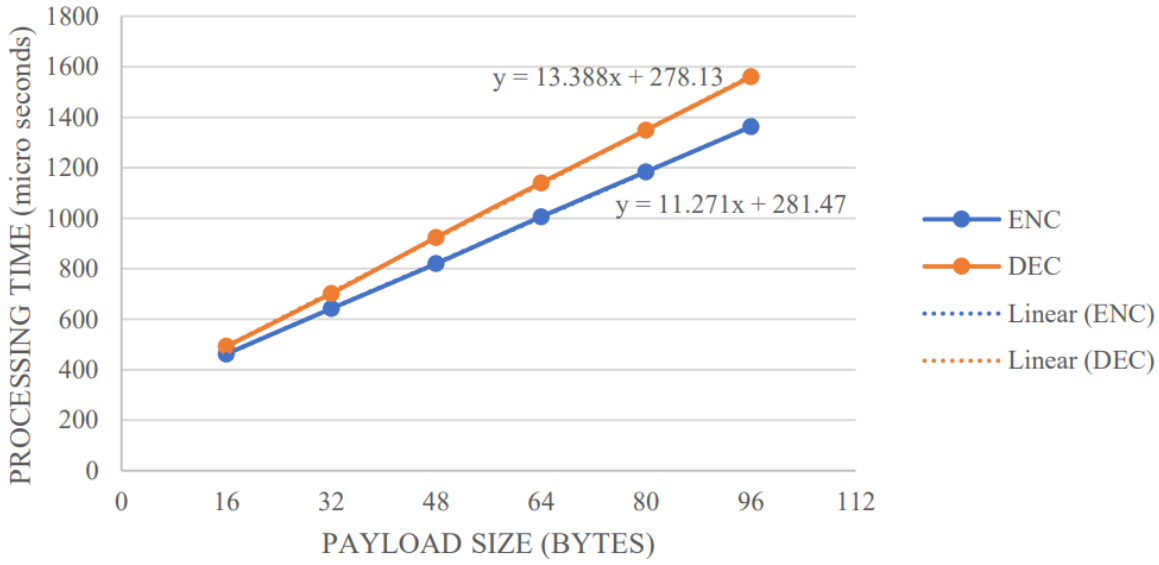
Fuente: Tomado de Nanabayin Menyah (2017).

Figura 71. Resultado del análisis del rendimiento de la carga útil cifrada.



Fuente: Tomado de Nanabayin Menyah (2017).

Figura 72. Tiempo medio de procesamiento para el análisis de cifrado y descifrado.



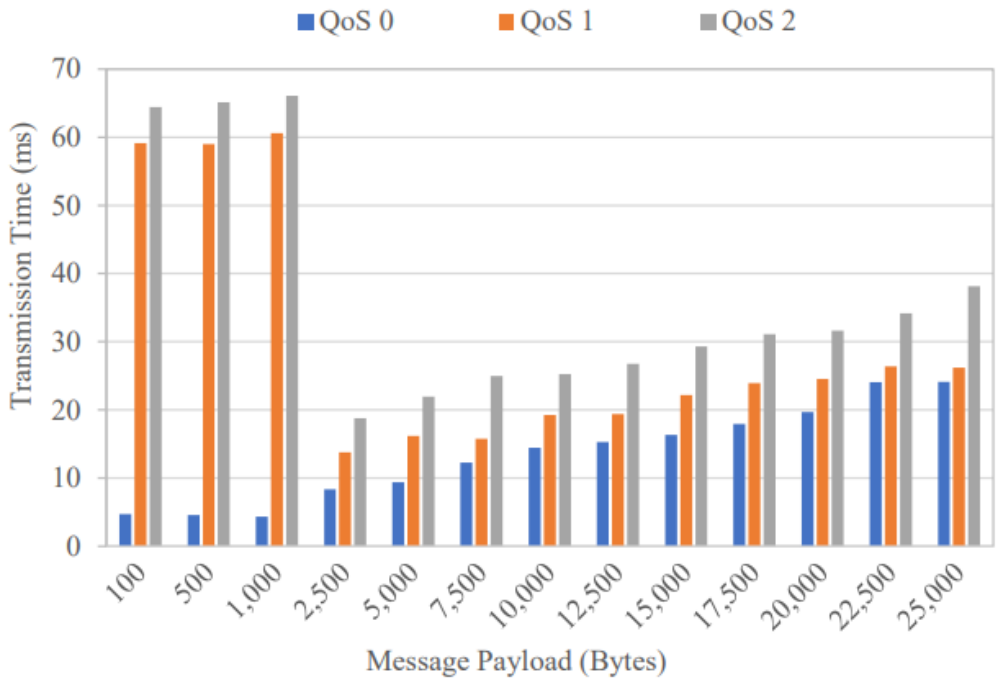
Fuente: Tomado de Nanabayin Menyah (2017).

7.20. Anexo 20

En las figuras 73, 74 y 75 se muestran los resultados obtenidos del articulo Performance Evaluation of Different Raspberry Pi Models as MQTT Servers and Clients (2022) los cuales describen el impacto del nivel de calidad QoS en el tiempo de transmision de un mensaje PUBLISH en MQTT.

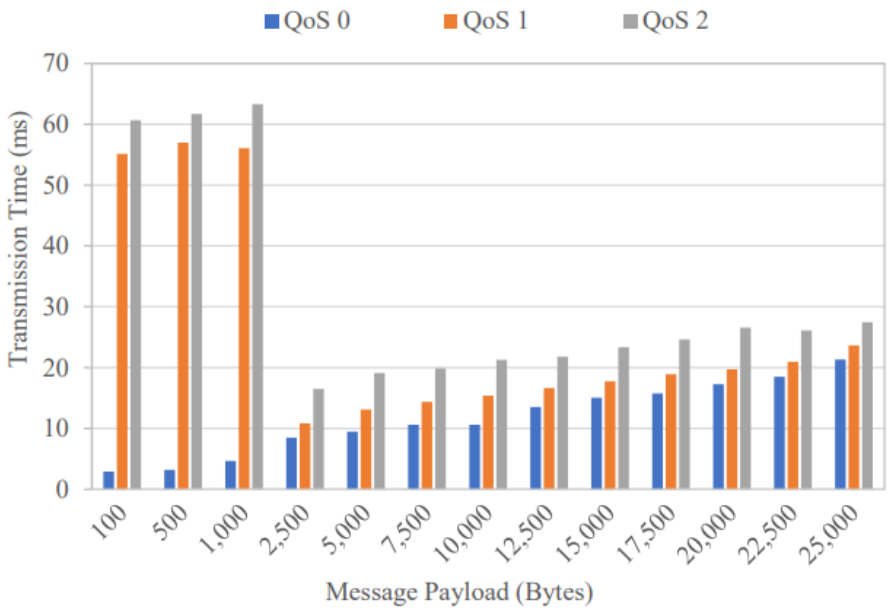
En este experimento, se utilizaron dispositivos Raspberry Pi como clientes para publicar los mensajes, mientras que una PC actuó como intermediario. Los resultados del experimento utilizando RPi Zero W, RPi Zero 2 W y RPi 3B. El tamaño del mensaje MQTT PUBLISH varió de 100 a 25.000 bytes. El ancho de banda WiFi se estableció en un máximo de 145 Mbps en el enrutador utilizando la banda de 2.4 GHz (N. Ford, Gamess, & Ogden, 2022).

Figura 73. Tiempo de transmisión cuando se usa RPi Zero W como clientes con variación de QoS



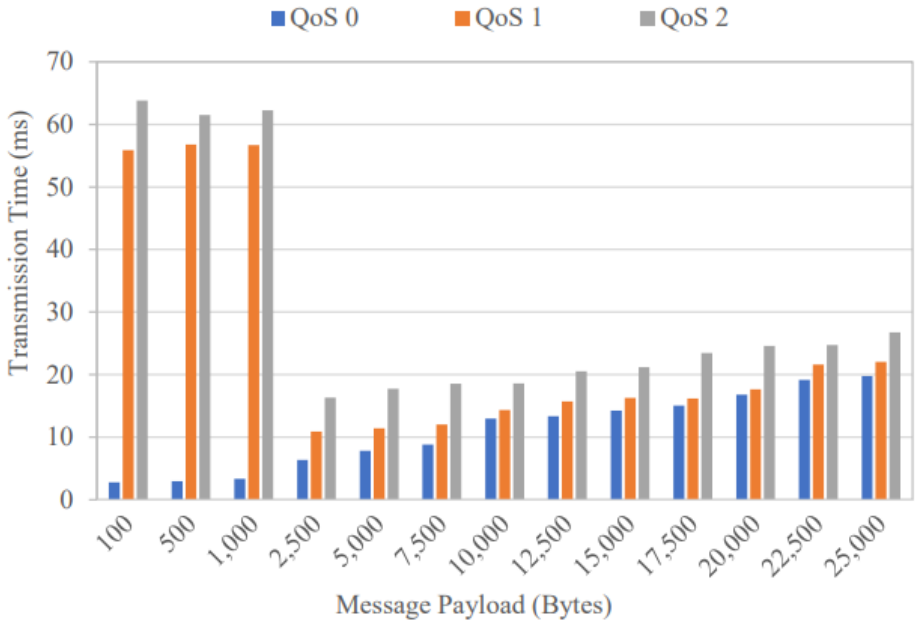
Fuente: Tomado de N. Ford, Gamess, & Ogden (2022).

Figura 74. Tiempo de transmisión cuando se usa RPi Zero 2 W como clientes con variación de QoS



Fuente: Tomado de N. Ford, Gamess, & Ogden (2022).

Figura 75. Tiempo de transmisión cuando se usa RPi 3B como clientes con variación de QoS



Fuente: Tomado de N. Ford, Gamess, & Ogden (2022).

Autor: Jorge Hernan Ticona Sanga

Correo electrónico: hernan_davion7@hotmail.com

Numero de celular: 78879989

2023-TTES-1173-D-1

DIRECCIÓN DE DERECHO DE AUTOR
Y DERECHOS CONEXOS
RESOLUCIÓN ADMINISTRATIVA NRO. 1-2791/2023
La Paz, 4 de Octubre del 2023

VISTOS:

La solicitud de Inscripción de Derecho de Autor presentada en fecha **27 de Septiembre del 2023**, por **JORGE HERNAN TICONA SANGA** con **C.I. Nº 7058775 LP**, con número de trámite **DA 1446/2023**, señala la pretensión de inscripción de la Tesis de Post-Grado titulada: **"DESARROLLO DE UNA TÉCNICA DE EVALUACIÓN DE DESEMPEÑO DE PROTOCOLOS PARA LA CAPA DE APLICACIÓN UTILIZADOS EN LA INTERNET DE LAS COSAS"**, cuyos datos y antecedentes se encuentran adjuntos y expresados en el Formulario de Declaración Jurada.

CONSIDERANDO

Que, en observación al Artículo 4º del Decreto Supremo Nº 27938 modificado parcialmente por el Decreto Supremo Nº 28152 el *"Servicio Nacional de Propiedad Intelectual SENAPI, administra en forma desconcentrada e integral el régimen de la Propiedad Intelectual en todos sus componentes, mediante una estricta observancia de los regímenes legales de la Propiedad Intelectual, de la vigilancia de su cumplimiento y de una efectiva protección de los derechos de exclusiva referidos a la propiedad industrial, al derecho de autor y derechos conexos; constituyéndose en la oficina nacional competente respecto de los tratados internacionales y acuerdos regionales suscritos y adheridos por el país, así como de las normas y regímenes comunes que en materia de Propiedad Intelectual se han adoptado en el marco del proceso andino de integración"*.

Que, el Artículo 16º del Decreto Supremo Nº 27938 establece *"Como núcleo técnico y operativo del SENAPI funcionan las Direcciones Técnicas que son las encargadas de la evaluación y procesamiento de las solicitudes de derechos de propiedad intelectual, de conformidad a los distintos regímenes legales aplicables a cada área de gestión"*. En ese marco, la Dirección de Derecho de Autor y Derechos Conexos otorga registros con carácter declarativo sobre las obras del ingenio cualquiera que sea el género o forma de expresión, sin importar el mérito literario o artístico a través de la inscripción y la difusión, en cumplimiento a la Decisión 351 Régimen Común sobre Derecho de Autor y Derechos Conexos de la Comunidad Andina, Ley de Derecho de Autor Nº 1322, Decreto Reglamentario Nº 23907 y demás normativa vigente sobre la materia.

Que, la solicitud presentada cumple con: el Artículo 6º de la Ley Nº 1322 de Derecho de Autor, el Artículo 26º inciso a) del Decreto Supremo Nº 23907 Reglamento de la Ley de Derecho de Autor, y con el Artículo 4º de la Decisión 351 Régimen Común sobre Derecho de Autor y Derechos Conexos de la Comunidad Andina.

Que, de conformidad al Artículo 18º de la Ley Nº 1322 de Derecho de Autor en concordancia con el Artículo 18º de la Decisión 351 Régimen Común sobre Derecho de Autor y Derechos Conexos de la Comunidad Andina, referentes a la duración de los Derechos Patrimoniales, los mismos establecen que: *"la duración de la protección concedida por la presente ley será para toda la vida del autor y por 50 años después de su muerte, a favor de sus herederos, legatarios y cesionarios"*.

Que, se deja establecido en conformidad al Artículo 4º de la Ley Nº 1322 de Derecho de Autor, y Artículo 7º de la Decisión 351 Régimen Común sobre Derecho de Autor y Derechos Conexos de la Comunidad Andina que: *"...No son objeto de protección las ideas contenidas en las obras literarias, artísticas, o el contenido ideológico o técnico de las obras científicas ni su aprovechamiento industrial o comercial"*.

Que, el artículo 4, inciso e) de la ley 2341 de Procedimiento Administrativo, instituye que: *"... en la relación de los particulares con la Administración Pública, se presume el principio de buena fe. La confianza, la cooperación y la lealtad en la actuación de los servidores públicos y de los ciudadanos ..."*, por lo que se presume la buena fe de los administrados respecto a las solicitudes de registro y la declaración jurada respecto a la originalidad de la obra.

POR TANTO

El Director de Derecho de Autor y Derechos Conexos sin ingresar en mayores consideraciones de orden legal, en ejercicio de las atribuciones conferidas

RESUELVE:

INSCRIBIR en el Registro de Tesis, Proyectos de Grado, Monografías y Otras Similares de la Dirección de Derecho de Autor y Derechos Conexos, la Tesis de Post-Grado titulada: **"DESARROLLO DE UNA TÉCNICA DE EVALUACIÓN DE DESEMPEÑO DE PROTOCOLOS PARA LA CAPA DE APLICACIÓN UTILIZADOS EN LA INTERNET DE LAS COSAS"**, a favor del autor y titular: **JORGE HERNAN TICONA SANGA** con **C.I. Nº 7058775 LP**, quedando amparado su derecho conforme a Ley, salvando el mejor derecho que terceras personas pudieren demostrar.

Regístrese, Comuníquese y Archívese.

Abg. Carlos Alberto Soruco Arroyo
DIRECTOR DE DERECHO DE AUTOR
Y DERECHOS CONEXOS
SERVICIO NACIONAL DE PROPIEDAD INTELECTUAL

V°B°
Abg. Maribel Rivera
Apoyo Dirección
SENAPI

CASA/mxoq
c.c.Arch.

Oficina Central - La Paz
Av. Montes, Nº 515, entre Esq. Uruguay y C. Batallón Illimani.
Telfs.: 2115700
2115276 - 2115251

Oficina - Santa Cruz
Av. Uruguay, Calle prolongación Quijarro, Nº 29, Edif. Bicentenario.
Telfs.: 3121752 - 72042936

Oficina - Cochabamba
Calle Bolívar, Nº 737, entre 16 de Julio y Antezana.
Telfs.: 4484403 - 72042957

Oficina - El Alto
Av. Juan Pablo II, Nº 2560 Edif. Multicentro El Ceibo Ltda. Piso 2, Of. 58, Zona 16 de Julio.
Telfs.: 2141001 - 72043029

Oficina - Chuquisaca
Calle Kilómetro 7, Nº 366 casi esq. Urriolagoitia, Zona Parque Bolívar.
Telf.: 72005873

Oficina - Tarija
Av. La Paz, entre Calles Gro Trigo y Avaroa Edif. Santa Clara, Nº 243.
Telf.: 72015286

Oficina - Oruro
Calle 6 de Octubre Nº 5837 entre Ayacucho y Junín, Galería Central, Of. 14.
Telf.: 67201288

Oficina - Potosí
Av. Villazón entre calles Wenceslao Alba y San Alberto, Edif. AM. Salinas Nº 242, Primer Piso, Of. 17.
Telf.: 72018160

www.senapi.gob.bo