Performance analysis of communication protocols for Internet of Things platforms

Jhon Ramirez
Departamento de Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia, Colombia
jhframirezgu@unal.edu.co

César Pedraza
Departamento de Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia, Colombia
capedrazab@unal.edu.co

***Abstract*—The Internet of Things (IoT) allows transform current Internet communication in an automatically way through Machine-to-Machine (M2M) communication. The IoT probably has become one of the most popular networking concepts that has the potential to bring out many benefits. However, IoT has brought some communications challenges. In the next three years, it is expected that more than 20 billion IoT devices to be connected, thus, communication protocols must allow massive multicasting, low overhead, and simplicity to operate on small platforms. This paper presents a comparative study of the main communication protocols that allows the implementation of an IoT platform, to determine their computational load, overhead and network bandwidth. Communication protocols such as MQTT, AMQP, CoAP and XMPP were tested to determine CPU processing, transferred bytes and sent packets for different communication scenarios.**

***Index Terms*—Internet of Things, Internet de las Cosas, IoT, MQTT, CoAP, AMQP, protocolos de comunicación.**

I. Introducción

El Internet de las Cosas (IoT, por sus siglas en inglés) es un sistema de dispositivos de computación interrelacionados, máquinas mecánicas y digitales, objetos, animales o personas que tienen identificadores únicos y la capacidad de transferir datos y comunicarse entre sí a través de una red sin requerir interacción de humano a humano, o humano a máquina[10]. Ahora bien, una cosa (thing), en el Internet de las cosas, puede ser una persona con un implante de monitor de corazón, una mascota con un sensor de rastreo, un automóvil que tiene sensores incorporados para alertar al conductor cuando la presión de los neumáticos es baja, o cualquier otro objeto natural o artificial al que se le puede asignar una dirección IP y con la capacidad de transferir datos a través de una red, es decir, las aplicaciones son infinitas[11].

También se resalta, que actualmente el Internet de las cosas (IoT) y sus protocolos están entre los temas más altamente financiados en la industria y la academia ya que actualmente se encuentra en la parte superior de su ciclo de madurez, lo que implica que una gran cantidad de dinero está siendo invertido en estas tecnologías por la industria. La rápida evolución de la Internet móvil, la fabricación de mini-hardware (miniaturización de los chips), la microcomputación y la comunicación máquina a máquina (M2M) han permitido el surgimiento de las tecnologías IoT[12], [16].

978-1-5386-1060-2/17/31.00©2017$IEEE$

Las tecnologías IoT permiten que las cosas, puedan oír, ver, pensar, o actuar, permitiéndoles comunicarse y coordinarse con otras cosas para tomar decisiones que pueden ser tan críticas como salvar vidas o edificios. Transforman las ”cosas” de la computación pasiva y la toma de decisiones individuales para comunicarse activamente y colaborar para tomar una sola decisión crítica. [12]Los sensores integrados, la comunicación ligera y los protocolos de Internet permiten que IoT proporcione su importancia, sin embargo, imponen muchos desafíos e introducen la necesidad de estándares especializados y protocolos de comunicación[10].

Ahora bien, actualmente existen varios protocolos de comunicación orientados a la comunicación entre máquinas (M2M) y los dispositivos con recursos limitados, entre ellos se destacan, MQTT (Message Queue Telemetry Transport), CoAP (Constrained Application Protocol) y AMQP (Advanced Messaging Queuing Protocol), entre otros. Lo anterior permite plantear el siguiente problema: debido a la variedad de protocolos existentes, ¿Cuál es el mejor protocolo de comunicación para la implementación de una plataforma de internet de las cosas (IoT)?, lo que puede ser determinado a partir de sus costos de implementación, basado en la cantidad de bytes transferidos y la cantidad de paquetes enviados.

La estructura del artículo está organizada de la siguiente manera: en la segunda parte se tiene una breve descripción de los protocolos de comunicación más usados en el desarrollo e implementación de plataformas de IoT. En la tercera parte se encuentra el diseño y ejecución del experimento. En cuarto lugar se aprecia el análisis de los resultados obtenidos. En último lugar, las conclusiones.

II. Protocolos de comunicación en IoT

A continuación se presenta una breve descripción de los protocolos más utilizados en la implementación de plataformas de Internet de las cosas (IoT).

A. Message Queue Telemetry Transport (MQTT)

MQTT es un protocolo de mensajería que fue presentado por primera vez por Andy Stanford-Clark de IBM y Arlen Nipper de Arcom en 1999 y fue estandarizado en 2013[2]. MQTT tiene como objetivo conectar dispositivos embebidos y redes mediante aplicaciones y middleware[4]. MQTT utiliza

el patrón publish/subscribe para proveer flexibilidad y simplicidad de implementación[6]. También, MQTT es adecuado para dispositivos con recursos limitados que usan enlaces poco fiables o de bajo ancho de banda[7]. MQTT está construido en el tope del protocolo TCP, y entrega mensajes mediante tres niveles de calidad de servicio (QoS)[4], [7]. MQTT simplemente consiste en tres componentes: subscriber, publisher y broker. Un dispositivo interesado debe registrarse como subscriber en topics específicos si desea ser informado por el broker cuando los publishers publican topics de interés. Después de esto, el publisher transmite la información a las entidades interesadas (subscribers) a través del broker[4], [7], [8]. Por lo tanto, el protocolo MQTT representa un protocolo de mensajería ideal para las comunicaciones IoT y M2M[3] y es capaz de proveer direccionamiento en redes vulnerables y de bajo ancho de banda para dispositivos pequeños, económicos, de baja energía, y de poca memoria[7].

B. Advanced Messaging Queuing Protocol (AMQP)

AMQP es un protocolo de la capa de aplicación para IoT que se enfoca en ambientes orientados a mensajes. AMQP requiere de un protocolo de transporte confiable como TCP para intercambiar mensajes[1], [13]. La comunicación en este protocolo es manejada por dos componentes principales: el área de intercambio (Exchanges) y las colas (Messages queues). Los exchanges son usados para direccionar los mensajes a las messages queues apropiadas. El direccionamiento entre los exchanges y las messages queues está basado en algunas reglas y condiciones predefinidas[13]. Los mensajes pueden ser almacenados en messages queues y entonces ser enviados a los receptores. Más allá de este tipo de comunicación punto a punto, AMQP también soporta el modelo de comunicación publish/subscribe.

Este protocolo permite que las aplicaciones envíen y reciban mensajes. En este sentido, es como la mensajería instantánea o correo electrónico. Donde AMQP difiere enormemente es que permite especificar los mensajes que se recibirán y de dónde proceden y cómo se hacen los compromisos con respecto a la seguridad, la confiabilidad y el rendimiento[9].

C. Constrained Application Protocol (CoAP)

Este protocolo es un protocolo de la capa de aplicación para aplicaciones de IoT, y fue desarrollado por el grupo de trabajo Constrained RESTful Environments (CoRE) de la IETF[3], [7]. CoAP define un protocolo de transferencia web basado en REpresentational State Transfer (REST) en el tope de las funcionalidades de HTTP[2], [8]. REST permite a los clientes y servidores exponer y consumir servicios web como el protocolo Simple Object Access Protocol (SOAP), pero de una forma más sencilla usando Uniform Resource Identifiers (URI) como sustantivos y los métodos de HTTP get, post, put, delete como verbos[5], [14]. A diferencia de REST, CoAP usa el protocolo UDP[6], [7], [8], [13], [14] (REST usa TCP) por defecto, lo que lo hace más adecuado para las aplicaciones de IoT. Además, CoAP modifica algunas funcionalidades de HTTP para cumplir con los requerimientos de IoT, tales como, bajo consumo de energía, y operación en presencia de enlaces con pérdidas y ruidos[5].

CoAP es un protocolo de software pensado para ser usado en dispositivos electrónicos muy simples, permitiéndoles comunicarse interactivamente sobre Internet. Está particularmente orientado a pequeños dispositivos con recursos limitados de energía, cómputo, y capacidades de comunicación, tales como, sensores, switches, válvulas, y componentes similares que necesitan ser controlados o supervisados remotamente, a través de redes estándar de Internet, mediante el uso de interacciones RESTful[2], [4], [5].

D. Extensible Messaging and Presence Protocol (XMPP)

XMPP es un protocolo de comunicaciones que fue diseñado inicialmente para mensajería instantánea y el intercambio de mensajes entre aplicaciones. Fue estandarizado por la Internet Engineering Task Force hace más de una década, por lo tanto, es bien conocido y se ha probado que es altamente eficiente sobre internet[13]. Actualmente es muy usado para conversaciones, llamadas de voz y video, y telepresencia entre múltiples partes. XMPP permite a los usuarios comunicarse entre ellos enviando mensajes a través de internet, no importa que sistema operativo estén usando. XMPP permite a las aplicaciones de mensajería instantánea conseguir: autenticación, control de acceso, medición de privacidad, hop-by-hop, cifrado extremo a extremo y compatibilidad con otros protocolos [15].

Recientemente, ha sido usado para aplicaciones de Internet of Things (IoT). Este uso del mismo estándar es debido a su uso del lenguaje XML, el cual, lo hace fácilmente extensible. Este protocolo soporta las arquitecturas de Publisher/Subscriber y Request/Response. Está diseñado para aplicaciones en tiempo real y, así soportar eficientemente pequeños mensajes de baja latencia[13].

E. Data Distribution Service (DDS)

DDS es un protocolo publish/subscribe para comunicación en tiempo real de máquina a máquina (M2M) que ha sido desarrollado por el Object Management Group (OMG). En contraste con otros protocolos de aplicación que usan el modelo publish/subscribe como MQTT y AMQP, DDS se basa en una arquitectura sin broker y usa multidifusión para conseguir una excelente QoS (Quality of Service) y una alta confiabilidad a sus aplicaciones. Su arquitectura publish/subscribe sin broker se adapta bien a las condiciones de tiempo real para comunicaciones IoT y M2M. La arquitectura DDS define dos capas: Data-Centric Publish-Subscribe (DCPS) y Data-Local Reconstruction Layer (DLRL). DCPS es responsable de la entrega de información a los suscriptores. DLRL, por otro lado, es una capa opcional y sirve como una interface a las funcionalidades de DCPS. Facilita el comportamiento de datos distribuidos entre objetos distribuidos[6], [13].

III. Experimentos

A continuación se presentan los experimentos realizados. Por un lado, se seleccionan las métricas que se van a analizar para determinar cuál es el mejor protocolo, y por otro lado, se detalla la arquitectura del experimento.

A. *Selección de las métricas*

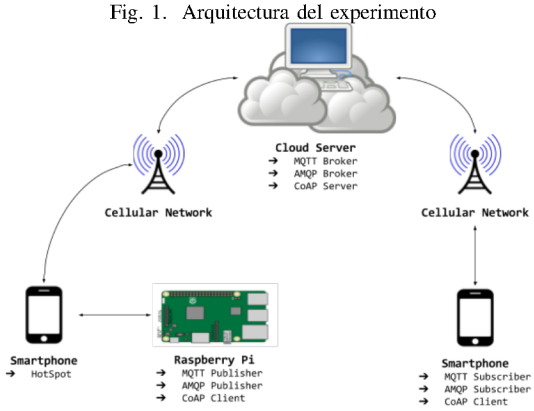
Las métricas que se pretenden calcular son:

- Tiempo de transmisión (Latencia): Tiempo total que tarda un paquete desde que sale del cliente origen hasta que llega al cliente destino.
- Tiempo de envío: Es el tiempo que tarda el paquete en salir del cliente de origen.
- Tiempo de CPU: Es el tiempo de procesamiento total de las máquinas (Raspberry PI y Dispositivo Móvil) donde se ejecutan los clientes cuando se envían y reciben los paquetes.
- Porcentaje de uso de CPU: Es el porcentaje de uso de los procesadores de las máquinas (Raspberry PI y Dispositivo Móvil) donde se ejecutan los clientes cuando se envían y reciben los paquetes.
- Bytes por transacción: Es la cantidad total de bytes que se envían en promedio para un tamaño de mensaje determinado.
- Paquetes enviados: Es la cantidad de paquetes que se envían en promedio para un tamaño de mensaje determinado.

B. *Arquitectura*

La arquitectura del experimento está conformada por un conjunto de componentes que se detalla a continuación (Figura 1).

Fig. 1. Arquitectura del experimento



- Cloud Server: Este es el servidor virtual donde se ejecuta el procesamiento de las peticiones de los clientes para cada uno de los protocolos, es quién recibe los paquetes de los clientes de origen y los envía a los clientes de destino. Es una instancia de Google Cloud Platform, una máquina *g1-small*, que consiste en un tipo de máquina de núcleo compartido, con una CPU virtual, 1.7GB de memoria RAM y 10GB de almacenamiento, ubicado en la zona *us-central1-a*. Ejecuta la distribución GNU/Linux Debian 8.7 (Jessie) con la versión de kernel 3.16.0-4.
- Smartphone (HotSpot): Es un dispositivo móvil, con acceso a internet proporcionado por la red celular de un proveedor de telefonía celular, que funciona como punto de acceso (Access Point) para los clientes (origen) que se ejecutan en la Raspberry Pi. Es un teléfono inteligente Motorola G 2a generación, modelo XT1068, con un procesador Quad-core 1.2 GHz Cortex-A7, 1GB de memoria RAM y 8GB de almacenamiento interno. Ejecuta el sistema operativo Android 6.0 Marshmallow.
- Raspberry Pi: Es un computador de placa reducida, donde se ejecutan los clientes de origen, para cada protocolo, que envían los paquetes al servidor para su procesamiento y posterior envío a los clientes destino. Es un computador de placa reducida modelo 3, con un procesador Quad-core ARMv8 a 1.2GHz de 64-bits, con 1GB de memoria RAM y una tarjeta micro SD de 8GB para almacenamiento interno. Ejecuta la distribución GNU/Linux para sistemas embebidos Arch Linux ARM con la versión de kernel 4.1.21-1.
- Smartphone (Client): Es un dispositivo móvil, con acceso a internet proporcionado por la red celular de un proveedor de telefonía celular, donde se ejecutan los clientes de destino, quienes reciben los paquetes enviados por el servidor. Es un teléfono inteligente Motorola G 1a generación, modelo XT1068, con un procesador Quad-core 1.2 GHz Cortex-A7, 1GB de memoria RAM y 8GB de almacenamiento interno. Ejecuta el sistema operativo Android 5.1 Lollipop.

Las siguientes son las librerías utilizadas para cada uno de los protocolos seleccionados:

- MQTT: Para el editor (publisher) se utilizó la librería paho v1.2.3 de Eclipse para el lenguaje de programación Python 2.7.3 para el suscriptor se utilizó la librería paho v1.1.0 de Eclipse como servicio de Android y para el servidor (broker) se utilizó la librería Mosquitto v1.4.11 de Eclipse.
- AMQP: Para el editor (publisher) se utilizó la librería pika v0.10.0 para el lenguaje de programación Python 2.7.3. para el suscriptor se utilizó el cliente. para el lenguaje de programación java en conjunto con la dependencia slf4j. del proyecto RabbitMQ; y para el servidor (broker) se utilizó la librería RabbitMQ v3.6.10 de RabbitMQ. [9].
- CoAP: Para el cliente de origen y el servidor se utilizó la librería desarrollada por Giacomo Tanganelli: CoAPthon v4.0.2 para el lenguaje de programación python 2.7.10. y para el cliente de destino se utilizó la librería Californium v1.0.0 de Eclipse.

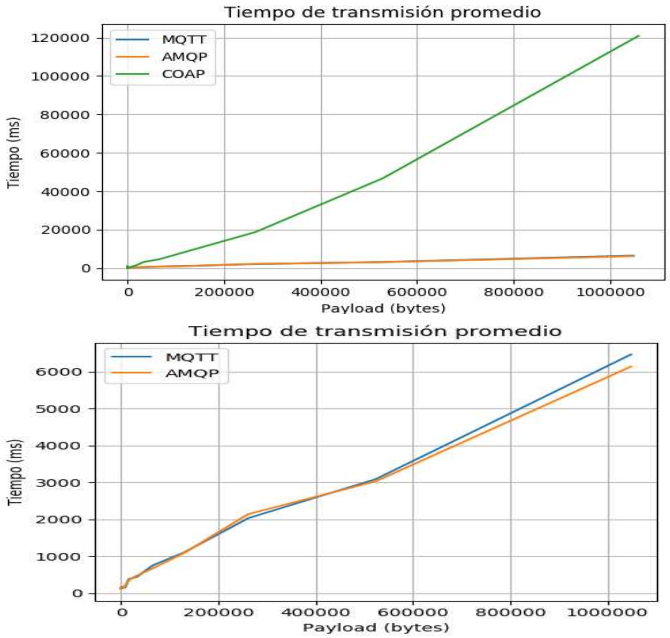
IV. Resultados

Todos los scripts necesarios para ejecutar el experimento, así como los archivos de resultados de las métricas y las gráficas de comparación se encuentran en el repositorio de GitHub http://github.com/RaiderColombia/proy_prot/.

El experimento se ejecutó con una configuración de 21 iteraciones, con un tamaño incremental de payload hasta un máximo de 2^{20} = 1.048.576 bytes (1MB), con 5 repeticiones de envío para cada mensaje, y con un retraso de 5 segundos entre cada mensaje. A continuación se presentan los resultados obtenidos, y su correspondiente análisis para cada una de las métricas seleccionadas.

En la figura 2 se puede apreciar una notable diferencia en el tiempo de transmisión de CoAP, ya que para transmitir un payload de 1MB tarda casi 2 minutos, mientras que MQTT y AMQP solo tardan 6.4 y 6.1 segundos respectivamente. Ahora bien, es evidente que mientras el payload sea menor de 1KB CoAP tarda en promedio 238ms, que es cerca del doble de los tiempos de MQTT, 126ms, y AMQP 125ms. Con esto, también se puede observar que estos dos últimos protocolos tienen un tiempo de transmisión muy parecido entre ellos, pero cabe resaltar que después de un payload de 512KB AMQP es mucho más rápido que MQTT.

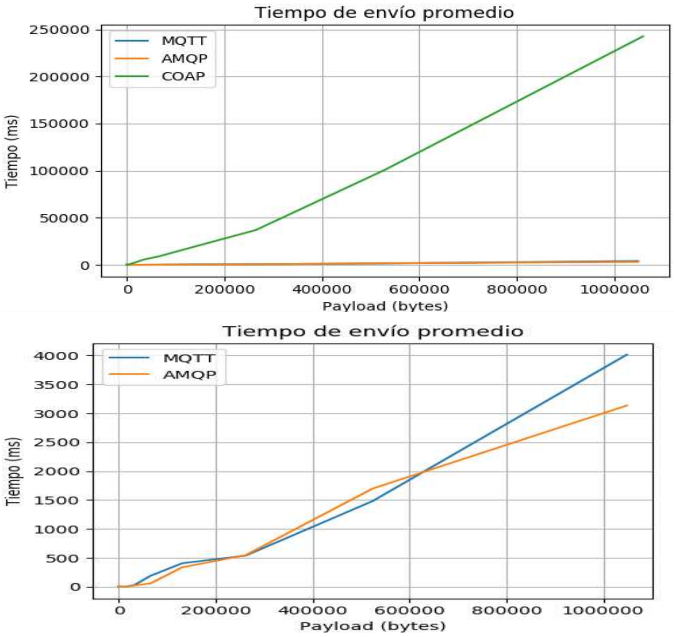
Fig. 2. Tiempo promedio de transmisión (Latencia)



A partir de la figura 3 se nota que CoAP es mucho más lento que MQTT y AMQP, además, también se puede ver que tarda mucho más tiempo enviando los paquetes que recibirlos, consume el doble de tiempo, esto ocurre porque el cliente de origen tarda más tiempo construyendo el paquete que se va a enviar, que el servidor en enviarlo al cliente de destino. En cuanto a la comparación entre MQTT y AMQP, es claro, que a partir de un payload mayor a 32KB AMQP es el protocolo más rápido de los tres, lo que significa que puede enviar los mensajes mucho más rápido, y esta diferencia se acentúa en la medida que el payload va creciendo, muestra de ello es que cuando el payload es de 1MB la diferencia con MQTT es de casi 1 segundo. Ahora bien, también se puede ver que con un payload menor a los 32KB el tiempo consumido en el envío de paquetes es muy similar con MQTT.

En la figura 4 se corrobora el pobre desempeño de CoAP, ya que se observa que el tiempo que tarda el proceso en ejecución en el procesador es mucho mayor que MQTT y AMQP, esto se debe a que debe construir y enviar una cantidad mayor de paquetes para lograr la meta del payload definido en la iteración, lo que requiere de mayor procesamiento. También,

Fig. 3. Tiempo promedio de envío



se puede ver en la figura 4 que cuando el payload es menor a 16KB, MQTT y AMQP tienen tiempos de procesamiento muy similares, aunque MQTT es más inestable que AMQP, es por ello que se pueden ver cambios tan variables en las primeras medidas. Cuando el payload es mayor a 16KB, es claro que AMQP tarda menos tiempo de procesador, lo que indica que es más eficiente que MQTT, ya que tarda un 75% menos ejecutándose en el procesador, lo que a su vez indica que también permite un ahorro en la energía del dispositivo donde se ejecute, ya que el consumo de energía y el procesamiento están directamente relacionados.

Como complemento a los tiempos de procesamiento, con el porcentaje de uso de CPU se aprecia que debido a que AMQP tarda menos tiempo en ejecución en el procesador, el porcentaje de uso del mismo es el menor de los tres protocolos, seguido por MQTT, y por CoAP en último lugar, el cual es el protocolo que más tiempo tarda en ejecución. Ver figura 5. Aunque cabe resaltar que inicialmente los tres protocolos tardan más tiempo del promedio cuando el payload es más pequeño, por ejemplo en AMQP en la primera iteración con un payload de 11 bytes es el mayor consumo de procesador que logra durante todo el experimento, lo mismo ocurre con MQTT en sus primeros tres iteraciones.

A continuación se presentan los resultados de las métricas de tamaño.

La figura 6 ilustra que cuando se trata de bytes transferidos hay una relación lineal, esto ocurre en los tres protocolos. Cuando se trata de MQTT, existe la siguiente relación, si el payload es menor a 138 bytes se adicionan 11 bytes, si el payload es mayor o igual a 138 bytes y menor a 16KB se adicionan 12 bytes, y cuando el payload es mayor a igual a 16KB pero menor a 65KB son 13 bytes los que se adicionan,

Fig. 4. Tiempo promedio de procesamiento

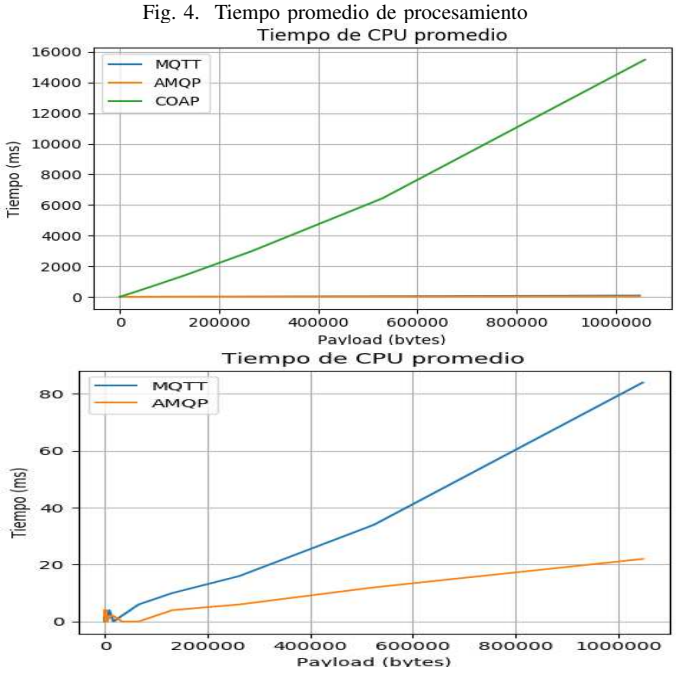
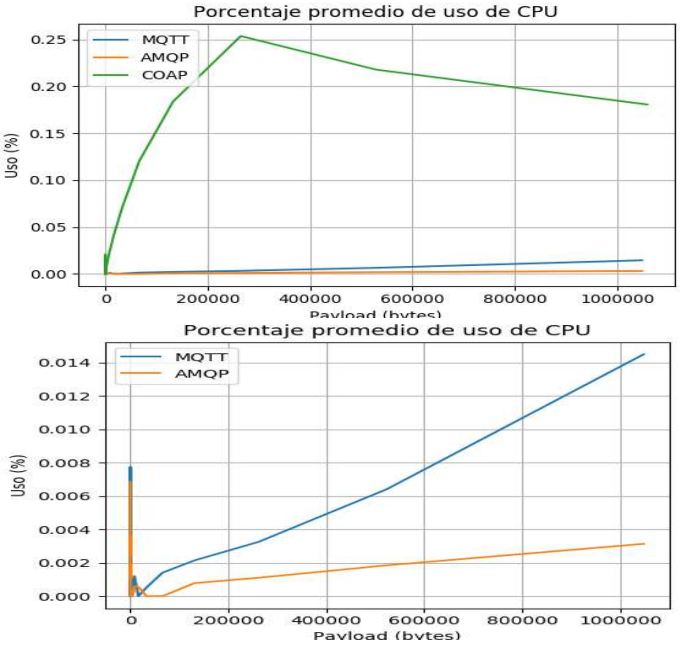


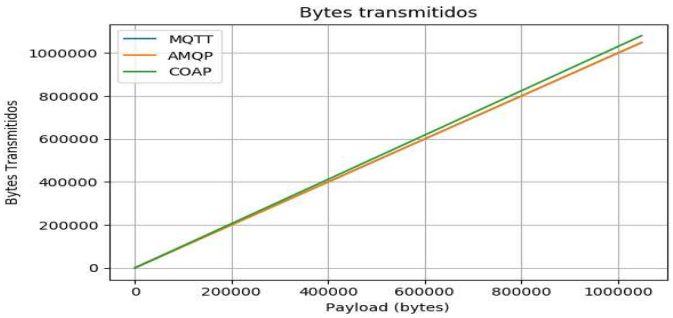
Fig. 5. Porcentaje promedio de procesamiento



por último si el payload es mayor que 65KB se adicionan un total de 68 bytes al payload generando así el tamaño total del paquete. En cuanto a AMQP, si el payload es menor o igual a 1KB se agregan 30 bytes, y si el payload supera 1KB pero es menor que 65KB entonces se adicionan 8 bytes, si el payload es mayor que 65KB y menor o igual a 132KB se agregan 45 bytes, luego si el payload es mayor que 132KB y menor que 1MB se adicionan nuevamente 8 bytes en total. Por último, para CoAP, se agregan 21 bytes si el payload es menor a 2KB, y si es mayor que 2KB se adiciona un incremento de bytes con una razón de cambio de 1,03027.

Lo anterior muestra que a mayor tamaño de payload, mayor cantidad de bytes se agregan al paquete, incrementando así su tamaño total.

Fig. 6. Total bytes transferidos

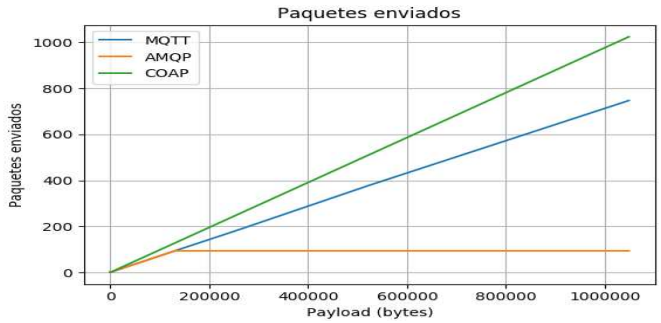


Como se puede observar en la figura 7, los tres protocolos envían un paquete si el tamaño del payload es menor o igual a 1KB, si el tamaño es mayor, cada protocolo hace una fragmentación y procede a enviar un número de paquetes de determinado tamaño para lograr transmitir la totalidad del mensaje.

En el caso de CoAP, debido a que el tamaño máximo del payload es de 65KB, se implementó un mecanismo que envía paquetes de 1KB hasta completar la totalidad del payload que se desea enviar, es por esta razón que el número de paquetes enviado se encuentra en potencias de 2, despues de un payload de 1KB.

En cuanto a MQTT hay un incremento en el número de paquetes enviado de acuerdo al tamaño del payload, a diferencia de AMQP quien envía una cantidad variable de paquetes si el payload es mayor a 1KB y menor a 132KB, ya que después de esta cantidad, se envía un valor constante de 94 paquetes, no importa el tamaño del payload.

Fig. 7. Número de paquetes enviados

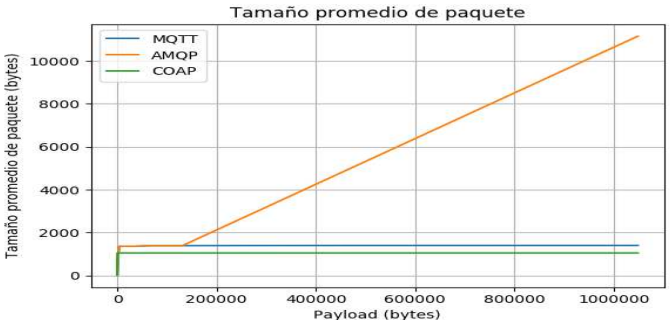


Las anteriores gráficas y tablas permiten apreciar que MQTT y CoAP envían un mayor número de paquetes que AMQP, pero dichos paquetes tienen un tamaño pequeño comparado con el payload, esto es debido a que estos dos

protocolos tienen un tamaño máximo para el payload, por un lado, MQTT puede enviar mensajes con un payload máximo de 256MB, y por otro lado, CoAP tiene un máximo de 65KB, lo que implica que deben controlar el máximo tamaño que puede llegar a tener el paquete.

En cuanto a AMQP, se puede apreciar que el tamaño promedio del paquete que se envía aumenta con respecto al tamaño del payload, esto sucede porque después de un payload mayor de 132KB el número de paquetes que se envía se vuelve constante, 94 paquetes, y para cubrir la totalidad del mensaje que se desea enviar, el protocolo aumenta el tamaño promedio del paquete. Ver figura 8.

Fig. 8. Tamaño promedio de paquete



V. Conclusiones

- MQTT y AMQP tienen un desempeño similar en cuanto a latencia, tiempos de envío, y bytes transferidos, cuando el tamaño del payload es pequeño, menos de 1MB. Es evidente que AMQP es ligeramente más rápido, y más eficiente, puesto que sus tiempos de procesamiento y porcentaje de uso del procesador son menores que los otros protocolos.
- Para completar el envío del payload deseado, AMQP envía pocos mensajes de gran tamaño, mientras que MQTT y CoAP envían un gran número de mensajes pequeños.
- Existen varias librerías para cada uno de los protocolos seleccionados, pero el desempeño de los mismos radica en la eficiencia y optimización de la implementación de dichas librerías.
- El modo observador de CoAP permite implementar una arquitectura Editor/Suscriptor, similar a la arquitectura de MQTT y AMQP, pero aunque se asemeja a dicha arquitectura, su desempeño no es muy eficiente.
- La plataforma Google Cloud Platform, y su instancia *g1-small* fue suficiente para ejecutar el experimento, además de permitir realizar las pruebas necesarias para determinar los parámetros adecuados del experimento. Debido a que no se sobrepasaron las cuotas definidas para la versión gratuita, no se incurrió en gastos que incrementará el valor de la ejecución del experimento. Aunque cabe resaltar, que si se despliega una plataforma de IoT, es necesario tener en cuenta la cantidad de peticiones que se realizan, y el tiempo entre cada una de ellas, ya que ello podría incrementar los costos del servidor.
- Después de ejecutar el experimento, se puede concluir que el protocolo con mejor desempeño es AMQP, ya que sus tiempos de transmisión, envío, y procesamiento son menores que los otros protocolos, lo que lo hace un protocolo más rápido. También, sobresale por el porcentaje de uso de procesador, lo que demuestra que es un protocolo más eficiente y que a su vez favorece la disminución en el consumo de energía de los dispositivos, que es una característica importante de los sistemas limitados en recursos. Por último, el hecho de que el protocolo pueda enviar mayor cantidad de información con un número menor de transacciones, permite que el costo para enviar un mensaje disminuya, puesto que al enviar menos paquetes, se requieren menos bytes adicionales al payload.

References

[1] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks," in 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), 2015, pp. 931–936.

[2] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A Comparison of Lightweight Communication Protocols in Robotic Applications," Procedia Computer Science, vol. 76, pp. 400–405, Jan. 2015. D. H. Mun, M. L. Dinh, and Y. W. Kwon, "An Assessment of Internet of Things Protocols for Resource-Constrained Applications," in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016, vol. 1, pp. 555–560.

[3] D. H. Mun, M. L. Dinh, and Y. W. Kwon, "An Assessment of Internet of Things Protocols for Resource-Constrained Applications," in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016, vol. 1, pp. 555–560.

[4] N. D. Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in 2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), 2013, pp. 1–6.

[5] W. Colitti, K. Steenhaut, N. D. Caro, B. Buta, and V. Dobrota, "Evaluation of constrained application protocol for wireless sensor networks," in 2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN), 2011, pp. 1–6.

[6] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," in 2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT), 2016, pp. 1–7.

[7] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," in 2015 18th International Conference on Intelligence in Next Generation Networks, 2015, pp. 70–75.

[8] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014, pp. 1–6.

[9] J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues, and S. Ullah, "Performance evaluation of RESTful web services and AMQP protocol," in 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN), 2013, pp. 810–815.

[10] S. Kraijak and P. Tuwanut, "A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends," in 11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015), 2015, pp. 1–6.

[11] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Communications Surveys Tutorials, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[12] G. Gardašević et al., “The IoT Architectural Framework, Design Issues and Application Domains,” Wireless Pers Commun, vol. 92, no. 1, pp. 127–148, Jan. 2017.
[13] Internet of Things Protocols and Standards. (2016). Cs.wustl.edu. Retrieved 7 November 2016, from http://www.cs.wustl.edu/~jain/cse570-15/ftp/iot_prot/index.html
[14] Joncas, R. (2016). MQTT and CoAP, IoT Protocols. Eclipse.org. Retrieved 7 November 2016, from http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php
[15] XMPP - Extensible Messaging And Presence Protocol. (2016). Xmpp.org. Retrieved 7 November 2016, from http://xmpp.org/
[16] Gartner’s 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business. (2014). gartner.com. Retrieved 7 November 2016, from http://www.gartner.com/newsroom/id/2819918