Trabajo Final Integrador

Recopilación y análisis de los protocolos de aplicación utilizados en IoT

para la obtención del título de

Especialista en Ingeniería en Sistemas de Información

Universidad Tecnológica Nacional
Facultad Regional Santa Fe

Autor: Lic. Emanuel Lucas Orzuza

Director: Mg. Ing. Aldo D. Sigura

Contenido

Introducción 4

Objetivos 7

Objetivo General 7

Objetivos Específicos 7

1- Modelos y protocolos a desarrollar 8

Arquitectura publicar / suscribir 10

Arquitectura cliente / servidor 11

2 - MQTT 12

Características 13

Arquitectura 15

Estructura de una trama MQTT 16

Aspectos relativos a la seguridad en MQTT 20

Autorizaciones 21

Integridad y privacidad del mensaje. Control de paquetes 21

3 - AMQP 22

Características 22

Arquitectura 24

Tipos de intercambiadores 25

Tramas en AMQP 26

Aspectos relativos a la Seguridad 29

4 - HTTP 31

Características 31

Arquitectura 33

RESTful APIs 34

Trama HTTP 36

Peticiones (Request) 37

Respuestas (Response) 39

Diferencias con HTTP/2 41

Aspectos relativos a la Seguridad 42

Autenticación HTTP 43

5 - CoAP 44
Características 44
Arquitectura 45
Arquitectura de COAP siguiendo el modelo cliente / servidor 46
Arquitectura de COAP siguiendo el modelo observador 48
Aplicación del modo observador 50
Tramas en CoAP 51
Opciones en la trama 53
Aspectos relativos a la Seguridad 54
6 – Análisis comparativo. Conclusiones 56
Arquitectura 56
Mensajes 56
Características de Servicio 57
Seguridad 58
Conclusiones 59
Bibliografía 61

Introducción

El término IoT (Internet de las Cosas) se refiere a escenarios en los que la conectividad de red y la capacidad de cómputo se extienden a objetos, sensores y artículos de uso diario que habitualmente no se consideran computadoras, permitiendo que estos dispositivos generen, intercambien y consuman datos con una mínima intervención humana (1).

Internet de las cosas se presenta como un gran avance que permite que diferentes productos de uso personal, electrodomésticos, automóviles y demás bienes, utilicen información proveniente de internet para su funcionamiento. Las proyecciones del impacto que tendrá IoT sobre Internet y la economía son impresionantes: los expertos estiman que existirán más de 35 mil millones de dispositivos en 2020 y más de 75 mil millones en 2025 (2), tal como se detalla en la **Figura 1**

IoT está cambiando la forma en la que los objetos y dispositivos que usamos a diario se comunican entre sí y, de manera consecuente, la forma en la que nosotros nos comunicamos con esos dispositivos y objetos. Existen miles de millones de dispositivos conectados, generando grandes volúmenes de información y presionando sobre las capacidades de las organizaciones y sus infraestructuras, para almacenar, procesar y analizar datos (3).

Estos miles de millones de dispositivos, provienen de numerosos fabricantes e interactúan bajo diferentes estándares y protocolos. La transmisión de datos desde y hacia los dispositivos IoT puede realizarse, entonces, de múltiples maneras pero para el usuario esto resulta transparente. El usuario final del dispositivo tampoco conoce (o debería conocer), si la transmisión de los datos se realiza de forma segura, manteniendo la confidencialidad e integridad. En IoT, cada objeto es accesible a través de internet y tiene la capacidad de comunicarse y procesar información. Esta información, muchas veces de gran caudal, puede ser de carácter confidencial, debiendo garantizar su seguridad**. La privacidad e integridad de la información juegan un rol fundamental en los dispositivos IoT.**

En búsqueda de ampliar el mercado y ser los primeros en lanzar un producto novedoso, muchas veces las empresas desarrolladoras de tecnología, prestan poca atención a la seguridad de la información que almacenan, procesan o transmiten sus dispositivos. Con el objeto de ser más competitivos, muchos desarrolladores de dispositivos IoT han admitido que han debido realizar campañas de mercadeo y vender, antes de poder adecuar los dispositivos y conexiones a las medidas de seguridad pertinentes (4). Como consecuencia, las aplicaciones y otros software que reside en los dispositivos IoT, se publican y salen a mercado muchas veces con potenciales vulnerabilidades, que son saneadas mediante parches o actualizaciones. Pero, según estudios (5), la poca cultura del usuario en lo que respecta a actualizaciones de dispositivos IoT (no tienen interiorizado como actualizar el software de una heladera, un televisor

o un enchufe, por ejemplo) provoca que existan millones de dispositivos activos vulnerables a ataques de algún tipo.

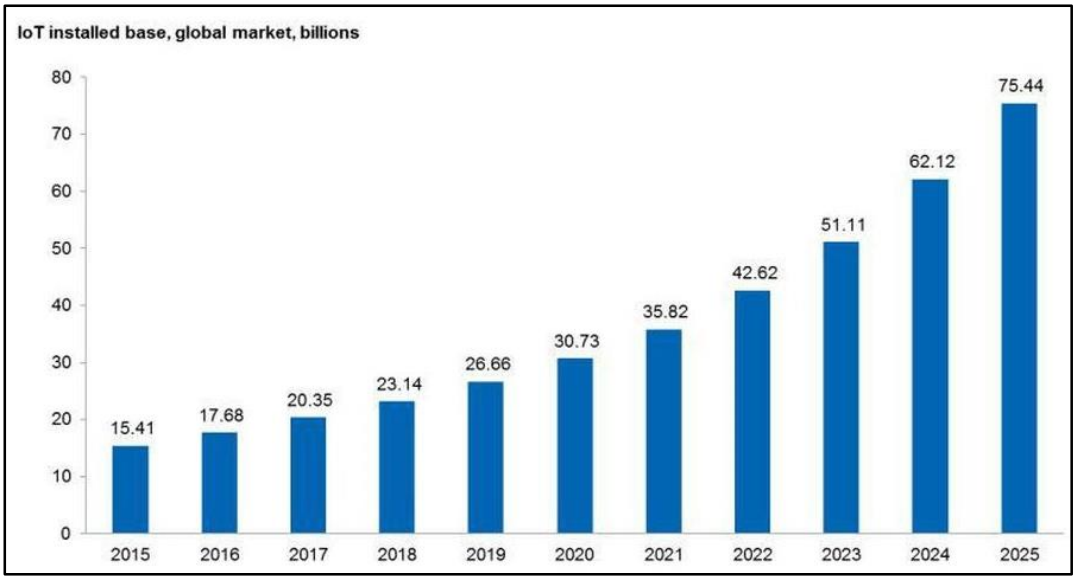


Figura 1. Expectativa de cantidad de dispositivos IoT, en miles de millones por año.

Fuente: IHS Tecnology

IoT introduce una amplia gama de nuevos riesgos y desafíos de seguridad para los propios dispositivos IoT, sus plataformas y sistemas operativos e incluso los sistemas a los que están conectados (debe tenerse en cuenta el uso de dispositivos IoT como canal de ataque). La empresa consultora y de investigación de las tecnologías de la información Gartner, publicó una encuesta donde afirma que más del 30% de los líderes de tecnología de las diferentes empresas que desarrollan dispositivos IoT, entendieron a la seguridad como barrera superior al éxito de IoT (6).

Además de los problemas de **seguridad** que pueden existir en el software que reside en los dispositivos IoT; es importante prestar atención a **la forma en la que estos dispositivos transmiten y reciben la información hacia y desde internet.** Un gran desafío que presenta IoT es la interoperabilidad: conectar dispositivos de la más diversa índole y naturaleza (desde electrodomésticos a vehículos, pasando por maquinas industriales), fabricados a su vez por miles de marcas diferentes (que muchas veces poseen sus propios estándares), junto a plataformas IoT que también manejan diversos esquemas y estándares de comunicación. Es aquí donde juegan un rol fundamental los protocolos, y es **donde este trabajo se centrará.**

El Comité de Arquitectura de Internet (IAB) dio a conocer un documento para guiar la creación de redes de objetos inteligentes (RFC 7452), que describe un marco de cuatro modelos de comunicación comunes que utilizan los dispositivos de la IoT. Estos son: comunicaciones

dispositivo a dispositivo, comunicaciones dispositivo a nube, comunicaciones dispositivos a puerta de enlace y un modelo de intercambio de datos a través de backend (7).

La importancia del desafío de interoperabilidad en IoT ha sido enfatizada tanto por el mundo académico como por la industria. Esta última intenta abordar los desafíos de interoperabilidad de IoT a través de la estandarización. Han surgido varios esfuerzos para establecer estándares para proporcionar interoperabilidad entre dispositivos de IoT, redes, servicios, formatos de datos propiedad de diferentes proveedores (8). La supervivencia y el crecimiento de los dispositivos IoT está íntimamente atado a su interoperabilidad: es necesario que todos los dispositivos hablen el mismo idioma

Figura 2. Problemas de seguridad ocultos en dispositivos IoT: software, hardware y comunicaciones.

Fuente: ArcServe.

Fundamentando en la interoperabilidad, que es la base y el gran desafío de los miles de millones de dispositivos IoT, tomando como referencia lo descripto en la RFC 7452 (7), y teniendo como premisa la seguridad de la información transmitida, almacenada y procesada por estos, es que **este trabajo indagará sobre los protocolos más usados en la capa de aplicación de los dispositivos IoT**.

Objetivos

Objetivo General

Recopilar y analizar los protocolos de aplicación que más utilizan los dispositivos IoT, clasificando y especificando ventajas y desventajas de cada uno.

Objetivos Específicos

- Recopilar y analizar los protocolos IoT del tipo publicar / suscribir y del tipo cliente / servidor.

- Relevar y analizar la tendencia en el uso de los protocolos de aplicación en dispositivos IoT

- Examinar e investigar la forma en la que los protocolos de la capa de aplicación aseguran la información que transmiten.

- Brindar recomendaciones relativas a seguridad de la información y transmisión de la información segura en el uso de dispositivos IoT

1- Modelos y protocolos a desarrollar

La capa de aplicación define las aplicaciones de red y los servicios de Internet estándar que puede utilizar un usuario (9). Estos servicios utilizan la capa de transporte para enviar y recibir datos. Existen varios protocolos de capa de aplicación, que podemos agrupar en dos categorías principales: protocolos **cliente/servidor** *(client/server)* y protocolos **publicar/suscribir** *(publish/subscribe)*.

Los protocolos cliente/servidor requieren que el cliente se conecte al servidor y realice solicitudes. En este modelo, el servidor tiene los datos y responde a los pedidos del cliente; esto requiere que el cliente conozca el servidor de antemano y sea capaz de conectarse.

En los protocolos publicar/suscribir los dispositivos se conectan a un "tópico" de un gestor intermediario y publican la información. Los consumidores se pueden conectar al gestor y suscribirse a los datos del tópico

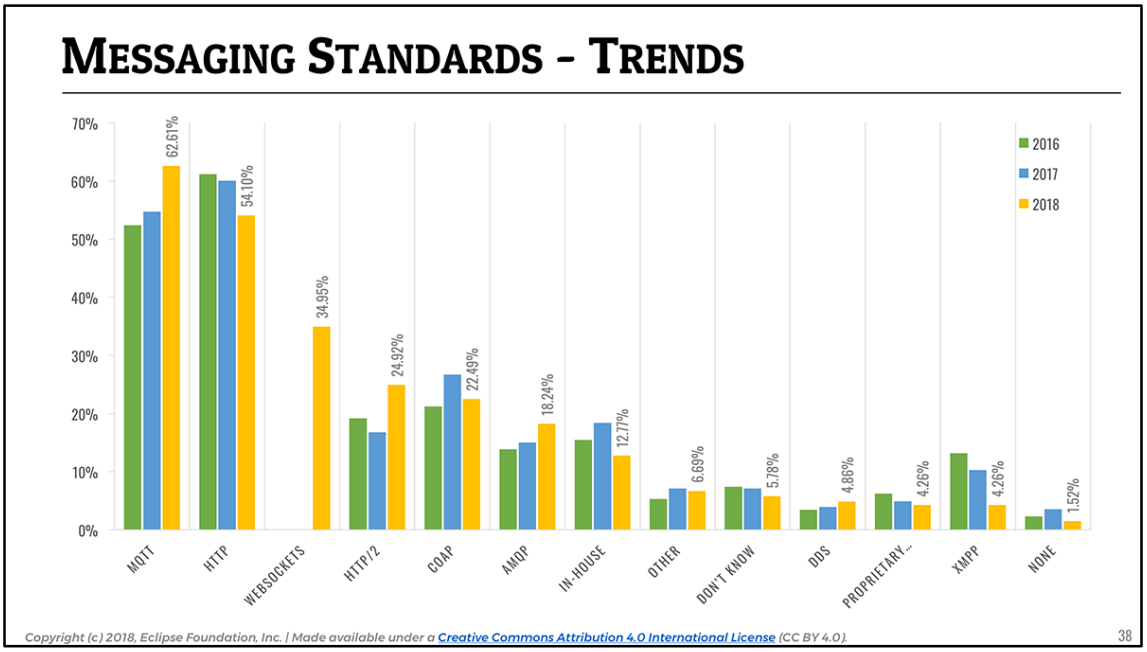


Figura 3. Protocolos de mensaje utilizados por IoT. **Fuente**: Eclipse Foundation.

Como puede observarse en la **figura 3** en IoT se utilizan variedades de protocolos, que incluyen del tipo publicar/suscribir (como MQTT) y del tipo cliente/servidor (como HTTP). Es menester indagar sobre las características y beneficios de cada uno de los protocolos de aplicación utilizados: el modo en el que transfieren la información, autenticación, cifrado, datos consumidos y demás información relevante para su uso en dispositivos IoT para poder realizar un análisis certero.

Dado la relevancia y el masivo uso que tienen algunos protocolos, y tomando como referencia las evaluaciones de las diferentes compañías que brindan, desarrollan o prestan servicios de IoT, este documento se interesara en recopilar información relevante a los protocolos **MQTT**, **HTTP**, **CoAP**, **AMQP**, en su uso para dispositivos IoT.

A continuación se brinda una breve introducción sobre el modo de funcionamiento de los protocolos *pub/sub* y *request/response*

Arquitectura publicar / suscribir

En las aplicaciones IoT, que fundamentalmente están basadas en la nube, los componentes del sistema necesitan proporcionar información a otros componentes a medida que suceden eventos. *La mensajería asincrónica es una forma eficaz de desacoplar a los remitentes de los consumidores y evitar bloquear al remitente para que espere una respuesta*

Un modelo de este tipo, tiene las siguientes características (10):

- Un canal de mensajería de entrada utilizado por el remitente. El remitente empaqueta los eventos en mensajes, mediante un formato de mensaje conocido, y envía estos mensajes a través del canal de entrada. El remitente en este patrón también se denomina publicador.
- Un canal de mensajería de salida por consumidor. Los consumidores se conocen como suscriptores.
- Un mecanismo para copiar cada mensaje del canal de entrada a los canales de salida para todos los suscriptores interesados en ese mensaje. Esta operación la controla típicamente un intermediario (*broker*)

En un modelo *pub / sub*, cualquier mensaje publicado sobre un tema es recibido inmediatamente por todos los suscriptores del mismo. Los modelos de publicación / suscripción se pueden usar en **arquitecturas controladas por eventos**, o para **desacoplar aplicaciones** con el objetivo de **aumentar el rendimiento, la confiabilidad y la escalabilidad** (11). Múltiples clientes pueden conectarse a intermediarios (*brokers*) y subscribirse a temas en los que están interesados. (12)

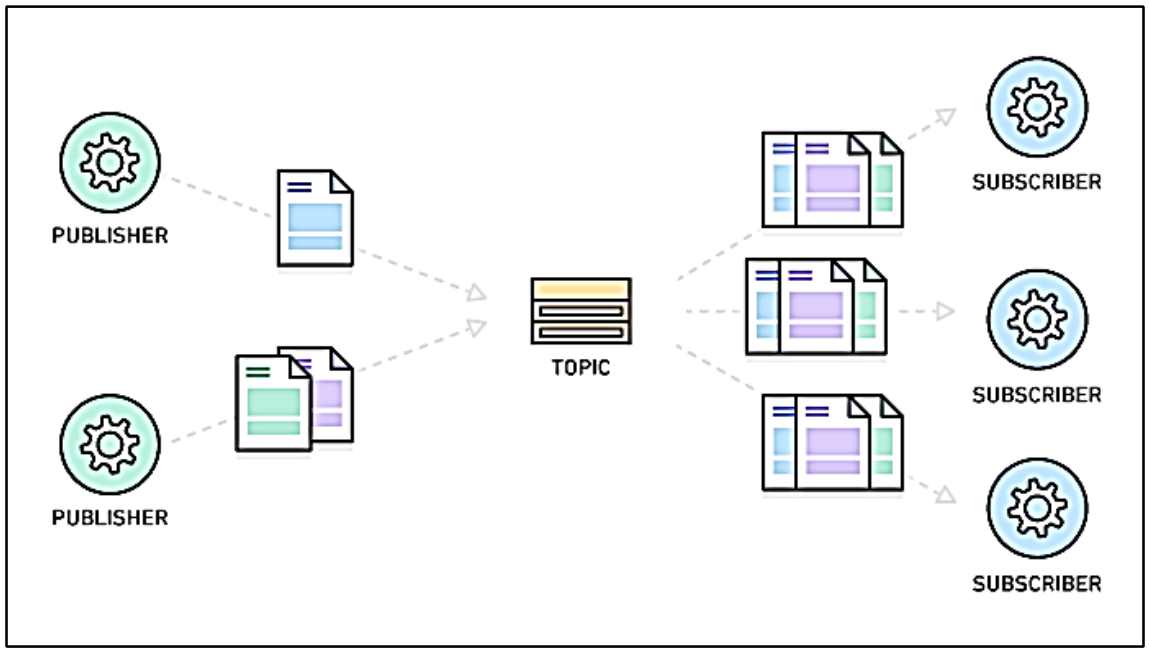


Figura 4: Modelo publicar / suscribir. **Fuente**: amazon.com

Arquitectura cliente / servidor

La arquitectura cliente-servidor es un modelo de diseño en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones al servidor, quien le da respuesta (13).

Un modelo de este tipo, tiene las siguientes características:

- Hay una relación muchos a uno entre los clientes y un servidor. Los Clientes generalmente inician un diálogo mediante la solicitud de un servicio. Los Servidores esperan pasivamente por las solicitudes de los clientes
- Encapsulación de servicios: El servidor es un especialista, cuando se le entrega un mensaje solicitando un servicio, él determina cómo conseguir hacer el trabajo. Los servidores se pueden actualizar sin afectar a los clientes en tanto que la interfaz pública de mensajes que se utilice por ambos lados, permanezca sin cambiar
- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes. Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
- Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente

En un modelo *request / response (o solicitud / respuesta)*, como el que representa una arquitectura cliente/ servidor, todos los controles están centralizados: accesos, recursos e integridad de datos almacenados son controlados por el servidor, que los entrega a solicitud del cliente; siempre y cuando este cuente con la debida autorización de acceso.

La escalabilidad es otra de las ventajas del modelo, debido a que la capacidad de almacenamiento, procesamiento o transferencia puede aumentar por separado para clientes y servidores. La congestión del trafico siempre ha sido un problema en el paradigma cliente/servidor (13); cuando una gran cantidad de clientes envían peticiones simultaneas al mismo servidor, este deberá tener la capacidad de responderlas o gestionarlas.

El modelo cliente/servidor, ha ido evolucionando, brindando soluciones a la congestión, generando alternativas de escalamiento horizontal, aumentando recursos y *hardenizando* las estructuras; esto ha posibilitado brindar servicios a clientes de la más variada índole, desde PCs de escritorio hasta lámparas inteligentes o sensores.

2 - MQTT

MQTT, acrónimo de *Message Queuing Telemetry Transport*, es un protocolo maquina a máquina (*M2M*), según define la RFC7452 (7), y fue diseñado como un transporte de mensajería extremadamente ligero; útil para conexiones con ubicaciones remotas donde se requiere pequeños fragmentos de código y/o el ancho de banda es reducido (14). Es uno de los principales protocolos *pub/sub* que utilizan los dispositivos IoT, tal como se puede observar en la **figura 5**.

Inicialmente, MQTT fue inventado y desarrollado por IBM a finales de los 90. Su aplicación original era vincular sensores en oleoductos de petróleo a satélites. Se trata de un protocolo de mensajería con soporte para la comunicación asíncrona entre las partes.

El protocolo MQTT es un estándar ISO en su versión 3.1.1 (año 2016) y un estándar OASIS en su versión 5, desde el año 2019 (15).

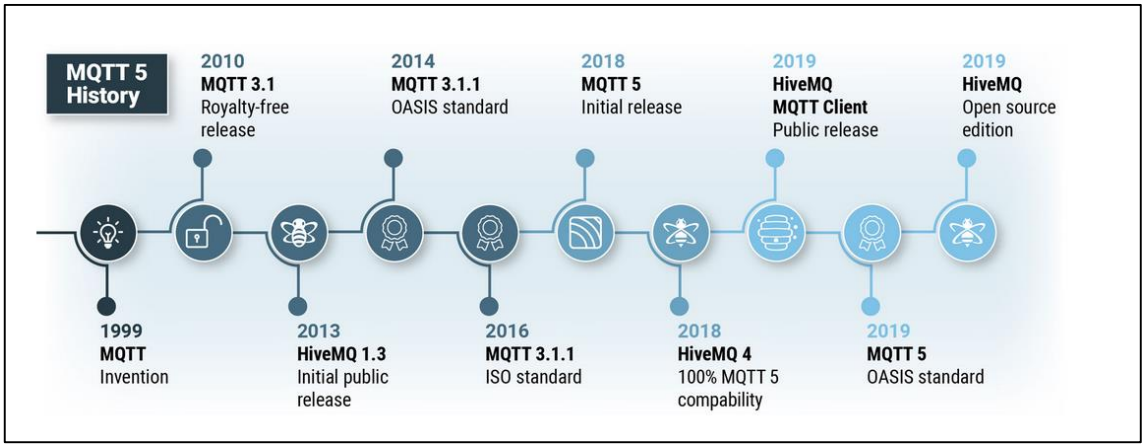


Figura 5: Evolución del protocolo MQTT. Fuente https://www.hivemq.com/blog/mqtt5-essentials-part1-introduction-to-mqtt-5/

MQTT es un protocolo maduro que ha estado impulsando muchos SCADA y aplicaciones de IoT durante años. Aunque una primera versión del protocolo fue creada a finales de la década de 1990, no fue hasta 2010 que una primera versión libre de MQTT fue lanzada al público, lo que sirvió como base para la especificación MQTT 3.1.1 estandarizada y abierta en 2014 (estándar OASIS)

Características

MQTT ofrece las siguientes características distintivas

- Es un protocolo de publicación/suscripción. Además de proporcionar distribución 'de uno a muchos', la publicación/suscripción desacopla las aplicaciones. Ambas funciones resultan útiles en aplicaciones que tengan muchos clientes. *Un solo cliente MQTT puede actuar como publicador y suscriptor para una comunicación bidireccional y desacoplada*
- No depende en modo alguno del contenido del mensaje.
- Utiliza TCP/IP en su capa de transporte, que se caracteriza por ser *confiable, de entrega de datos ordenada y sin errores.*
- Es económico en la forma en que gestiona el flujo de mensajes en la red (la cabecera de longitud fija tiene sólo 2 bytes de longitud y se minimizan los intercambios de protocolo para reducir el tráfico en la red).

MQTT lleva integrado en modo nativo la noción de calidad de servicio QoS (16), tal cual se puede apreciar en el cuadro de abajo. Los nodos tienen la posibilidad de definir la calidad de envío del mensaje; según tres niveles establecidos:

- ***Como máximo una vez*** *(*QoS=0*)*: Los mensajes de entregan en base a los mejores esfuerzos de la red de Protocolo Internet subyacente. Se puede producir pérdida de mensajes: el mensaje se entrega como máximo una vez, o no se entrega. *En esta QoS, el mensaje no se almacena*, y es posible perderlo ante algún fallo o desconexión. El protocolo MQTT no requiere que los servidores reenvíen publicaciones con QoS = 0 a un cliente. Si el cliente está desconectado en el momento en que el servidor recibe la publicación, es posible que la publicación se descarte. Consiste, básicamente, en la metodología *fire and forget*

- ***Al menos una vez*** *(*QoS=1**):** Es el modo de entrega por defecto. Se asegura que los mensajes llegan, pero se pueden producir duplicados.
  El mensaje siempre se entrega al menos una vez. Si el remitente no recibe un acuse de recibo, el mensaje se envía de nuevo con la bandera DUP activada hasta que se reciba un acuse de recibo. *Como resultado, al receptor se le puede enviar el mismo mensaje varias veces y podría procesarlo varias veces*.
  El mensaje debe almacenarse localmente en el remitente y el receptor hasta que se procese.
  El receptor elimina el mensaje después de haberlo procesado; si el receptor es un broker, el mensaje es procesado mediante la publicación a sus suscriptores; si el receptor es un cliente, el mensaje se entrega a la aplicación del suscriptor.
  Una vez eliminado el mensaje, el receptor envía un acuse de recibo al remitente.

Después de haber recibido un acuse de recibo del receptor, el remitente elimina el mensaje.

- ***Exactamente una vez*** *(*QoS=2)**:** Se asegura que los mensajes llegan exactamente una sola vez. Al igual que en QoS=1, el mensaje es almacenado localmente en el remitente y el receptor hasta que se procese.
  QoS = 2 es el modo de transferencia más seguro, pero también el más lento. Se necesitan al menos dos pares de transmisiones entre el remitente y el receptor antes de que el mensaje se elimine del remitente. En el primer par de transmisiones, el remitente transmite el mensaje y el receptor reconoce que ha almacenado el mensaje. Si el remitente no recibe un acuse de recibo, *el mensaje se envía de nuevo con la bandera DUP activada hasta que se reciba un acuse de recibo*.
  En el segundo par de transmisiones, el remitente le dice al receptor que puede completar el procesamiento del mensaje, enviando un mensaje “PUBREL”. Si el remitente no recibe un acuse de recibo del mensaje "PUBREL", el mensaje "PUBREL" se envía de nuevo hasta que se reciba un acuse de recibo. El remitente borra el mensaje que guardó cuando recibe el acuse de recibo del mensaje "PUBREL", *teniendo certeza que el mensaje será procesado*.
  El receptor puede procesar el mensaje en la primera o segunda fase, siempre que no vuelva a procesar el mensaje. El receptor envía un mensaje de finalización al remitente de que ha terminado de procesar el mensaje.

QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	<=1
1	0	1	At least once	Acknowledged delivery	>=1
2	1	0	Exactly once	Assured delivery	=1

Tabla 1. Niveles de QoS. Fuente: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

Ultima voluntad y testamento

MQTT proporciona un mensaje de "última voluntad y testamento” (*last will and testament - LWT*) que puede almacenarse en el broker MQTT en caso de que un nodo se desconecte inesperadamente de la red. Este *LWT* retiene el estado y el propósito del nodo, incluidos los tipos de comandos que publicó y sus suscripciones. Si el nodo desaparece, el broker notifica a todos los suscriptores del *LWT* del nodo. Y si el nodo regresa, el broker le notifica su estado anterior. Esta característica es muy buena para redes con pérdida y para entornos escalables

Arquitectura

La arquitectura de MQTT sigue una topología de estrella, teniendo un nodo central que hace de servidor o "broker". El broker es el encargado de gestionar la red y de transmitir los mensajes. La comunicación se basa en "topics" (temas), que crea el cliente que publica el mensaje y los nodos que deseen recibirlo deben subscribirse a él. La comunicación puede ser de uno a uno, o de uno a muchos.

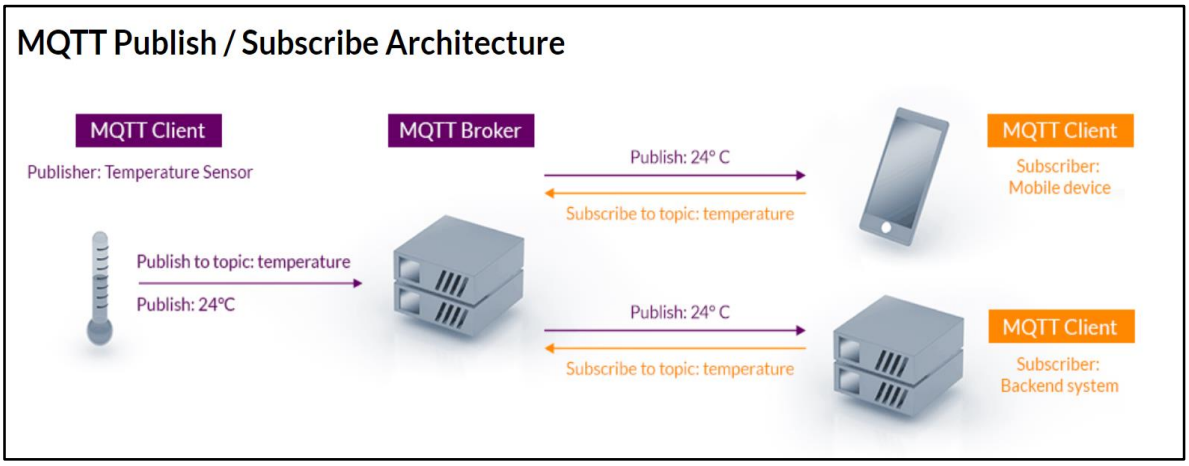


Figura 6: Arquitectura MQTT: Broker y clientes que pueden ser subscriptores o publicadores

Fuente: https://mqtt.org/

El bróker MQTT es el eje central del modelo. El servidor MQTT es responsable de la autenticación y autorización de los clientes MQTT que podrán convertirse en editores (publicadores) y / o suscriptores luego de que sean autenticados y autorizados.

Todos establecen una conexión con el bróker. El cliente que envía un mensaje a través del bróker se conoce como publicador. El broker o servidor filtra los mensajes entrantes y los distribuye a los clientes que están interesados (subscriptos) en cada uno de los tópicos o temas de los mensajes recibidos. Los clientes que se registran en el broker como interesados en tipos o temas específicos de mensajes se conocen como suscriptores.

Básicamente en el protocolo hay solo 3 actores involucrados: subscriptores, publicadores y el servidor o broker. El suscriptor puede también funcionar como publicador de otro tópico. Es decir que un mismo cliente puede ser subscriptor y publicador. Si la conexión del cliente suscriptor al broker se interrumpe; este almacenara los mensajes y los enviara al suscriptor cuando Evuelva a estar en línea. Por el contrario, si la conexión del cliente registrado como publicador se pierde o este se desconecta sin previo aviso, el broker puede cerrar la conexión y enviar a los suscriptores del tópico la notificación.

Estructura de una trama MQTT

MQTT es un protocolo binario donde los elementos de control son bytes binarios y no cadenas de texto. MQTT utiliza un formato de comando y reconocimiento de comando (*command and command acknowledgement*); eso significa que cada comando tiene un reconocimiento asociado, como se muestra en la figura 8.

El protocolo MQTT opera intercambiando una serie de paquetes de control siguiendo una estructura definida. Un paquete de control MQTT tiene 3 partes: Un encabezado fijo (*Fixed Header*), presente en todos los paquetes de control, un encabezado variable existente en algunos paquetes (*Variable/ Optional Header*), y una carga de datos útil (*Payload*) también opcional. Esta carga de datos puede ser de hasta 256 mbs (**en las implementaciones reales, raramente supera los 4 kb**).

Los paquetes de control, en su encabezado fijo, indican el tipo de paquete (CONNECT, CONNACK, PUBLISH, PUBAK, SUSCRIBE, UNSUSCRIBE, entre otros); indicando aquí también, el ***QoS*** que se utilizara cuando es del tipo PUBLISH (entre otras diferentes banderas de acuerdo al tipo de paquete de control). La Tabla 2 muestra el listado de todos paquetes de control que se pueden enviar, su dirección (cliente a servidor o servidor a cliente) y una breve descripción de cada uno. Son 14 tipos de mensajes posibles**)**

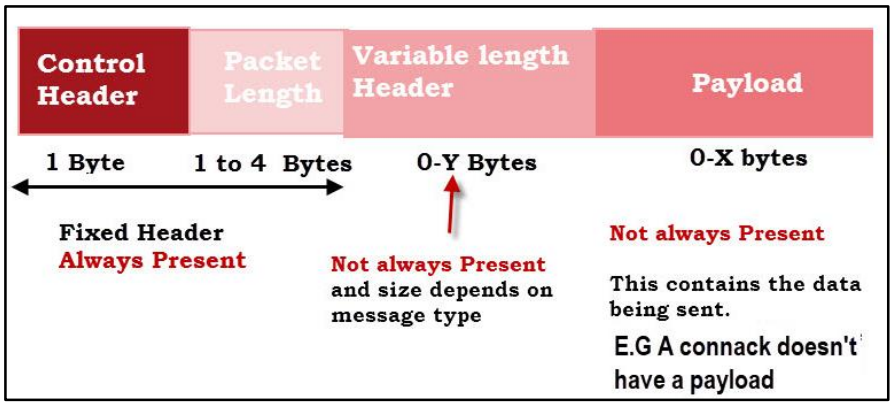


Figura 7-A. Paquete MQTT.

El **tamaño mínimo de un paquete MQTT es de 2bytes** (1byte del encabezado de control + 1 byte (mínimo) del largo del paquete). El **tamaño máximo son 256MB** (15). Un paquete DISCONNECT, por ejemplo tiene solo 2 bytes.

De los 8 bit (1 byte) del encabezado de control, los primeros 4 bits (los más significativos) e indican el tipo de mensaje. Los restantes 4 se utilizan como banderas de control. Existen 16 banderas posibles, pero en la realidad se usan muy pocas. Para el paquete PUBLISH, por ejemplo, pueden enviarse mediante flags, el QoS pretendido.

En la tabla 2 también pueden verse los valores asociados a cada uno de los paquetes de control.

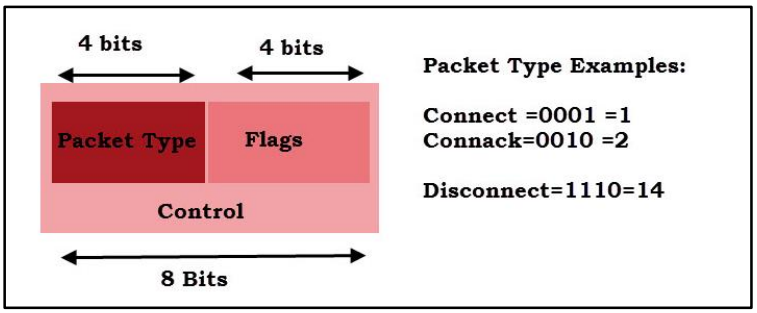


Figura 7-B. Encabezado de control de un paquete MQTT

MQTTT soporta BLOBS (Binary Large Object) de mensajes de hasta 256 MB de tamaño. El formato del contenido es específico de la aplicación. Las suscripciones de temas se realizan utilizando un par de paquetes SUBSCRIBE/SUBACK. La anulación de la suscripción se realiza de forma similar utilizando un par de paquetes UNSUBSCRIBE/UNSUBACK.

Durante la fase de comunicación, el cliente puede realizar operaciones de publicación, suscripción, cancelación (unsuscribe) y ping. La operación de publicación envía un bloque binario de datos, el contenido, a un topic definido por el *publisher*. La operación de ping al servidor del broker usando una secuencia de paquetes PINGREQ/PINGRESP, que se usa para saber si está viva la conexión. Esta operación no tiene otra función que la de mantener el vínculo "*vivo"* y asegurar que la conexión TCP no ha sido apagada.

La conexión MQTT se da siempre entre el cliente y el broker. Los clientes nunca se conectan directamente entre ellos. Para iniciar la conexión, el cliente envía un mensaje CONNECT al broker. Este responde con un mensaje CONNACK y un código que indica el estado. Cuando un publicador o suscriptor desea finalizar una sesión MQTT, envía un mensaje DISCONNECT al broker y, a continuación, cierra la conexión. Esto se denomina "graceful shutdown" porque le da al cliente la posibilidad de volver a conectarse fácilmente al proporcionarle su identidad de cliente y reanudar el proceso donde lo dejó.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

Tabla 2. Paquetes de Control de MQTT.
Fuente: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901019

Si la desconexión ocurre repentinamente sin tiempo para que un publisher envíe un mensaje DISCONNECT, el broker puede enviar a los suscriptores un mensaje del publisher que el broker ha almacenado previamente en caché (LWT). El mensaje, proporciona a los suscriptores instrucciones sobre qué hacer si el editor muere inesperadamente.

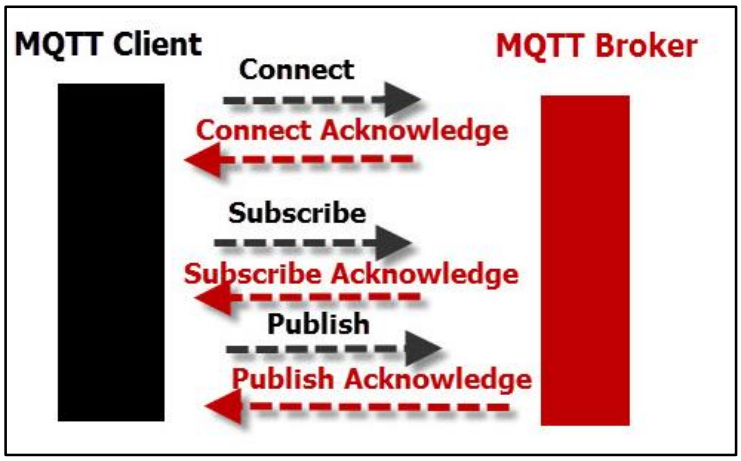


Figura 8. Comando y reconocimiento de comando en MQTT

Una vez iniciada la comunicación, el *broker* mantiene la conexión abierta hasta recibir un mensaje de DISCONNECT desde el cliente o hasta que la conexión se caiga por otro motivo.

Si el paquete CONNECT esta incorrecto (de acuerdo a la especificación del estándar) o pasa demasiado tiempo entre que se abre el socket de conexión y se envía el paquete, el broker puede dar por finalizada la conexión, asumiendo problemas de seguridad que podrían ralentizar la actividad del mismo (17).

A modo ejemplificatorio, se muestra un paquete CONNECT; que conformado de forma correcta debería tener los campos detallados en la Figura 8 (incluidos algunos otros opcionales)

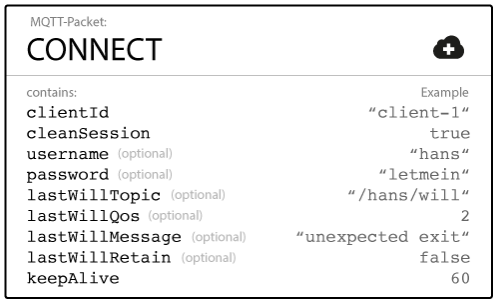


Figura 8. MQTT Connect Packet
Fuente: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/

El atributo *clientID* identifica cada cliente MQTT que se conecta al broker. Este ID debe ser único por cliente y broker, de manera de identificar unívocamente y poder conocer el estado de un determinado cliente, por ejemplo

El atributo cleanSession indica al broker si debe establecer una sesión persistente o no. En una sesión persistente (cleanSession=false) el broker almacena todas las subscripciones del cliente y los mensajes perdidos que tengan QoS 1 o 2. Si la sesión no es persistente el broker purga toda la información de sesión, inclusive de sesiones persistentes anteriores del mismo cliente.

El campo usuario y contraseña (username / password) puede utilizarse para autenticar y autorizar a un cliente. De todas formas, si estos campos no son encriptados de alguna manera, viajan por defecto en texto plano. Esto se analizará en el apartado de Seguridad.

La bandera *lastWillMessage* es parte la característica de "Ultima voluntad y testamento" especificado anteriormente y que permite notificar ante una desconexión inesperada y le da la posibilidad al broker de entregar al nodo su estado anterior en caso de una conexión.

Por último, el atributo *keepAlive* es un intervalo en segundos, especificado por el cliente al momento de establecer la conexión y que especifica el periodo de tiempo más largo que puede existir sin que se envíen mensajes entre el broker y el cliente.

Todas las banderas de conexión, en este caso, son parte del encabezado de tamaño variable (Variable Length header) y **se utilizan para indicar la presencia o ausencia** de nombre de usuario, password, banderas de mensaje **en el payload**.

Cuando el broker recibe un paquete CONNECT está obligado a responder con un paquete CONNACK, que incluye dos atributos: *sessionPresent* y *returnCode*. El primero indica al cliente si el broker ya tiene una sesión persistente disponible de interacciones anteriores. Si el cleanSession enviado por el cliente es true, el broker siempre contestara con un false en este campo. En cambio, en caso de que el cliente envie un cleanSession en false, el broker contestara de acuerdo al contenido (true si tiene información para ese cliente o false en caso contrario). Esta caracterisitca fue agregada en la versión 3.1 del estándar y **permite a un cliente conocer si debe subscribirse a un tópico o si los datos están en la sesión persistente en el broker**.

El segundo campo (returnCode) le indica al cliente si la conexión fue exitosa y puede tener varios valores: 0 (conexión aceptada), 1 (conexión rechazada, versión no aceptada del protocolo), 2 (conexión rechazada, identificador no valido), 3 (conexión rechazada, servidor no disponible), 4 (conexión rechazada, usuario o contraseñas incorrectas) y 5 (conexión rechazada, cliente no autorizado)

Como puede verse, en un simple paquete de conexión, hay varios atributos que ayudarían a mantener la seguridad de la información transmitida o almacenada cuando se utiliza MQTT. En al apartado siguiente, se pondrán en análisis aspectos relativos a la seguridad específicamente.

Aspectos relativos a la seguridad en MQTT

MQTT es usado comúnmente en ambientes de comunicación hostiles, donde numerosos factores de seguridad deben ser tenidos en cuenta. MQTT utiliza TCP sin cifrar, y **no es seguro "*listo para usar*".** Las implementaciones, con el fin de garantizar la seguridad de la información transmitida mediante este protocolo, deberían considerar efectuar medidas de seguridad en los procesos de **autenticación** de usuarios y dispositivos, de **autorización de accesos** a los recursos del servidor**,** y en los controles de **integridad y privacidad** de los paquetes de control y datos de la aplicación (18).

Podríamos dividir la seguridad de MQTT en múltiples capas. Cada una de ellas es factible de sufrir diferentes tipos de ataques: a nivel de red, a nivel de transporte o a nivel de aplicación. Y existen diferentes técnicas que el protocolo provee, facilita o recomienda para ampliar la seguridad en cada uno de estos casos.

MQTT tiene reconocimiento de sesión continua, dado que utiliza TCP/IP. **Por este motivo también permite (y recomienda ampliamente en la definición del estándar)** (18) **utilizar SSL/TLS para cifrar la comunicación y hacerla segura.**

Hay tres conceptos fundamentales en la seguridad de MQTT: identidad, autenticación y autorización. La **identidad** consiste en dar nombre al cliente que se va a autorizar y dar autorización. La **autenticación** consiste en probar la identidad del cliente y la **autorización**

consiste en gestionar los derechos que se otorgan al cliente (19). El protocolo provee mecanismos para llevar adelante estas premisas.

Autenticación y autorización entre clientes y servidores

El paquete CONNECT en MQTT contiene el campo usuario y contraseña, que pueden ser o no utilizados para identificar el cliente que desea conectarse al broker. Las implementaciones son capaces de usar estos campos con métodos de autenticación externa (como LDAP), utilizar otros métodos de autenticación (como los provistos por el sistema operativo) o desarrollar su propio mecanismo. Debe tenerse en cuenta que estas credenciales, a menos que se cifren mediante TLS, viajan en texto plano.

También es factible utilizar certificados SSL, utilizando TLS, para autenticar un cliente a un servidor, aunque debería analizarse la implementación en particular y el costo que conllevaría un certificado por cliente.

MQTT no es simétrico respecto a la seguridad. No proporciona ningún mecanismo para que el cliente autentique un servidor. Cuando se utiliza SSL el cliente puede utilizar los certificados provistos por el servidor para autenticarlo, identificando si el certificado es correcto y se corresponde con el broker.

Autorizaciones

Adicional a los controles de autenticación, MQTT también permite generar restricciones de autorización al uso de los recursos de un servidor, basándose en la información provista por el cliente, tales como el identificador de cliente, el nombre de host, o la dirección IP; otorgando permisos según corresponda.

Integridad y privacidad del mensaje. Control de paquetes

Independientemente del protocolo, las aplicaciones e implementaciones que utilizan MQTT, puede implementar *hashes* en sus mensajes de aplicación, que ayudarían al control de los paquetes a través de la red.

Utilizando TLS, podemos asegurarnos de la encriptación del contenido de los mensajes a través de la red. Independientemente del cifrado TLS en el envío del mensaje, cualquier aplicación que utilice MQTT puede cifrar el contenido de los mensajes de manera de proteger su contenido, tanto en la transmisión como durante su almacenamiento. Esto es una buena práctica recomendada por el estándar (20)

3 - AMQP

El Advanced Message Queuing Protocol (AMQP) es un estándar de código abierto que proporciona una interoperabilidad funcional completa para la comunicación de mensajes comerciales entre organizaciones o aplicaciones. El protocolo de comunicación fue propuesto por el JP Morgan en el año 2003; pero luego, firmas como Cisco y RedHat se unieron al proyecto ampliándolo en características.

En agosto de 2011, el grupo de trabajo AMQP había obtenido la membresía de OASIS, y dos meses después la versión 1.0 fue publicada. AMQP e**s un protocolo de mensajería corporativo diseñado para garantizar confiabilidad, seguridad, aprovisionamiento e interoperabilidad** (21).

En abril de 2014, OASIS AMQP recibió la aprobación de Norma Internacional ISO e IEC, bajo la designación ISO / IEC 19464[1]. A pesar de su desarrollo predominantemente dentro del sector financiero, AMQP ha tenido una amplia adopción como protocolo de comunicaciones de *M2M* dentro del entorno de IoT. AMQP es uno de los protocolos de capa de aplicación que ha tomado relevancia por ser utilizado en implementaciones destacadas y de gran dimensión como el proyecto Aadhaar de la India (22), considerado como uno de los las bases de datos biométricas más grandes del mundo. Otras notables implementaciones de AMQP contemplan instituciones como NASA, VMWare, Google e IBM (23)

AMQP brinda una amplia gama de características relacionadas con la mensajería, como una **cola confiable**, **mensajes de publicación y suscripción** basados en temas, **enrutamiento flexible** y **transacciones** (21).

Características

AMQP es un protocolo M2M (*machine to machine o máquina a máquina*) que funciona sobre la capa de aplicación; y que además fue diseñado como un protocolo de mensaje flexible, que contiene las siguientes características (24):

- Multiplexación de sesiones: El protocolo permite que múltiples sesiones sean iniciadas en una misma conexión TCP. Cada una de estas sesiones tiene flujo independiente y su propia secuencia de mensajes. Un cliente de aplicación puede, por ejemplo, recibir al mismo tiempo desde una cola y enviar a otra cola a través de la misma conexión de red

- Comunicación asíncrona, full dúplex: Dentro de una sesión los mensajes pueden fluir en ambas direcciones de manera independiente.

[1] https://www.iso.org/standard/64955.html ; Estándar ISO / IEC 19464. Consultado el 20 de Marzo de 2020

- Transferencia de Mensajes y acuse de recibo: AMQP define formalmente la semántica de transferencia de mensajes y acuse de recibo. Estas pueden ser
    - Como máximo una vez (*Settle Format)*
    - Al menos una vez (*Unsettle Format)*

- Seguridad de datos / información: AMQP provee la posibilidad de encriptar el contenido mediante TLS / SSL; utilizando encriptación y/o encriptación y autenticación mediante certificados 509.x. También provee mecanismos de autenticación como SASL (Simple Authentication and Security Layer)

- Control de flujo: AMQP tiene un control de flujo integrado en el protocolo para evitar sobrecarga de la aplicación por demasiados mensajes. Básicamente, se admiten dos modelos de control de flujo: control de flujo de sesión, que protege la infraestructura de un contenedor para que no se sobrecargue con demasiados mensajes, y control de flujo de enlace, que protege al mismo de la sobrecarga. Estos mecanismos de control de flujo permiten la creación de infraestructuras de mensajería robustas.

- Interoperabilidad: AMQP permite casi cualquier forma de mensajería, incluidas las colas de mensajes clásicas, roundrobin, store-and-forward y combinaciones de los mismos. Por ejemplo, algunos consumidores pueden obtener copias de mensajes mientras que otros se extraen directamente de la misma cola, todos usando diferentes filtros

- Metada: Como otros protocolos, AMQP permite que los mensajes tengan un encabezado; permitiéndose cualquier número de encabezados específicos de la aplicación en un mensaje. Estos encabezados se transportan por separado del cuerpo del mensaje (que puede estar cifrado, codificado y comprimido)

- Transaccional: AMQP admite diferentes casos de uso de reconocimiento y transacciones en las colas de mensajes. El protocolo también define, pero hace opcional, el soporte para transacciones distribuidas, como transacciones X / Open XA o MS DTC. (25)

- Extensible: AMQP ha definido explícitamente puntos de extensibilidad que permiten a cualquier proveedor generar extensiones que utilicen estándares acordados, de una manera compatible con, y utilizable por, implementaciones existentes.

Arquitectura

El objetivo principal de AMQP era reemplazar los sistemas de mensajería patentados y no interoperables existentes que obstaculizaban las comunicaciones en el sector financiero. Para la transmisión de los mensajes, el protocolo apuesta por los intermediarios: ***brokers de mensajería.*** En AMQP se proporciona un espacio de cola compartido o un intermediario; al que pueden acceder todas las aplicaciones que participan. Un mensaje se envía desde un publicador, a través de un *Exchange,* a la cola. Cada mensaje tiene una clave de enrutamiento que será utilizado por el *broker* para asignar el mensaje a una cola en particular, distribuir el mensaje a varias colas, duplicar cada mensaje en varias colas o distribuir varios mensajes a las colas definidas según la clave.

Los elementos básicos que componen una red AMQP pueden distinguirse en (26):

- ***Cola de mensajes (queue)*:** Es quien almacena los mensajes hasta que de manera segura puedan ser consumidos o procesados por una aplicación cliente (o múltiples aplicaciones clientes, consumidores de mensajes)
- ***Exchange*:** es también llamado el "intercambiador". Recibe los mensajes de quienes los publican (productores de mensajes, aplicaciones clientes) y los envía a las diferentes colas de mensajes según corresponda.
- ***Binding*:** El "*enlace*" define la relación entre una cola de mensajes y un intercambiador (*Exchange*) y proporciona criterios de enrutamiento de mensajes. Se construyen a partir de comandos de la aplicación cliente

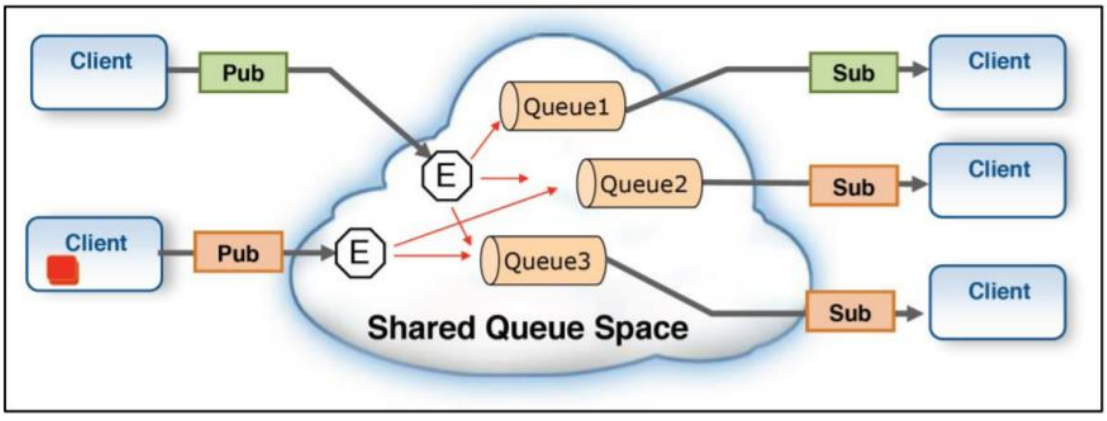


Figura 9. Flujo de un mensaje utilizando AMQP. Pub= Publicador, E = Exchange, Sub= Subscriptor
Fuente: Microsoft Conference. http://download.microsoft.com/documents/uk/msdn/events/sol/sol03.pdf

Las colas de mensajes funcionan como elementos de almacenamiento y distribución de mensajes: desde ellas los subscriptores consumen los mismos. Una cola de mensajes, además, tiene varias propiedades: pueden ser privadas o compartidas, permanentes o temporales, nombradas por el cliente o por el servidor, entre otras. Al seleccionar las propiedades

deseadas, se pueden generar colas de mensajes como entidades intermedias, según los criterios elegidos.

Cada cola tiene su propio nombre, que le identifica entre los demás participantes. Es posible decidir si un consumidor tiene que acusar recibo de un mensaje o es suficiente que el envío se realice correctamente. Si se elige la primera opción y el consumidor no envía ningún mensaje, el bróker intentará enviar el mensaje a otro consumidor o tratará de conectar con el mismo receptor una vez más. Sin embargo, si está activada la variante sin confirmación y el consumidor no reclama la entrega del mensaje, este se perderá.

También es posible que un cliente rechace la aceptación de un mensaje conscientemente. Esto puede ser útil cuando el procesamiento del mensaje no funciona. La respuesta del consumidor empuja al bróker, bien a borrar el mensaje por completo, bien a integrarlo de nuevo en la cola.

El ***Exchange*** acepta mensajes desde un productor (o publicador- *Figura 9*) y los rutea a colas de mensajes según criterios preestablecidos. Estos criterios se denominan "***bindings***" (o enlaces). Los *exchanges* (o intercambiadores) son motores de enrutamiento y correspondencia. Es decir, inspeccionan los mensajes y, utilizando sus tablas de enlace (*binding tables*), deciden cómo reenviar estos mensajes a colas de mensajes u otros intercambiadores. Los intercambiadores nunca almacenan mensajes. (27)

AMQP utiliza protocolos subyacentes compatibles con TCP / IP en los niveles de red y transporte, lo que proporciona interoperabilidad y confiabilidad, además de flexibilidad.

Tipos de intercambiadores

Los *exchanges* son los principales responsables de reenviar los mensajes recibidos a una cola u otro Exchange que se haya vinculado por adelantado de acuerdo con ciertas reglas. Este proceso del conmutador Excahnge se denomina enrutamiento. Hay tres implementaciones populares de enrutamiento AMQP: **Intercambio directo, Fanout Exchange** y **Topic Exchange**. Se debe prestar especial atención a qué tipo de reglas de "enrutamiento" debe tener el Exhange no se aplica en el protocolo estándar AMQP. Las populares y actuales reglas de reenvío de AMQP están desarrolladas por los productos de implementación de AMQP.(Esta es la razón por la cual los atributos relacionados con las reglas de enrutamiento y filtrado en los mensajes AMQP se almacenan en el área de propiedades de la aplicación).

A continuación se detalla cada uno de los tipos de intercambio que puede realizar AMQP mediante sus *exachanges* (28):

El primer tipo, el **intercambiador directo**, envía mensajes a un receptor concreto y trabaja para ello con claves de enrutamiento. Al mensaje se le transmite este tipo de clave. Una cola, a su vez, tiene una clave de vinculación. Esta identifica la cola frente al intercambiador. Si la clave de enrutamiento y la clave de vinculación son iguales, el mensaje se puede enviar a la

cola y con ello al receptor del mismo. También es posible que una cola tenga varias claves de vinculación y, por lo tanto, se pueda utilizar incluso para varias claves de enrutamiento. Por el contrario, varias colas también pueden compartir una clave de vinculación, que se conoce como vinculación múltiple. El intercambio multiplica el mensaje y lo envía a varios destinatarios.

El **intercambiador de abanico** (*fanout)* funciona de manera similar. Sin embargo, el bróker ignora en este caso la clave de enrutamiento por completo. En su lugar, *el intercambiador envía un mensaje a todas las colas disponibles y reproduce la información allí*. Por el contrario, el **intercambiador con un tema definido** (*topic exchange*), al igual que en un intercambiador directo, la clave de enrutamiento y las claves de vinculación se emparejan. Sin embargo, no debe existir una coincidencia exacta; en su lugar, se utilizan comodines. De esta forma se pueden preparar mensajes específicos para varias colas.

Debe tenerse en cuenta que en el modo de enrutamiento directo y el modo de enrutamiento por tema, si el conmutador de Exchange no encuentra ninguna cola que coincida con la clave de enrutamiento, el mensaje AMQP se descarta. (Solo las colas tiene la función de guardar mensajes, pero los Exchange no son responsables de hacerlo)

Una característica y ventaja de la arquitectura de AMQP, no menor, es que permite actuar como si el protocolo fuera cliente/servidor o pub/sub. Si se desea, se puede registrar a un tópico y se recibirán todas las novedades y mensajes publicados desde ese tópico. Por el contrario, si se desea no suscribir, se puede consumir los recursos de la cola cuando se considere necesario; actuando como cliente/servidor.

Tramas en AMQP

Un frame (o trama) es la unidad básica en AMQP. Una conexión está compuesta de una secuencia ordenada de frames. En este caso, el término orden hace referencia a que el último frame no debe llegar al receptor antes de que los demás frames hayan alcanzado previamente su objetivo. Cada frame se puede dividir en tres segmentos según la especificación del estándar (29) :

- **Header**: Encabezado obligatorio que tiene un tamaño de 8 bytes. Aquí se encuentra la información que determina el enrutamiento del mensaje.
- **Header extendido**: este ámbito es opcional y no tiene ningún alcance definido. Sirve para ampliar la información del header en un futuro.
- **Cuerpo**: en el cuerpo se encuentran realmente los datos a transmitir. El tamaño se puede seleccionar de forma arbitraria. Esta zona, sin embargo, puede dejarse en blanco, de forma que el frame solo sirve para mantener la conexión.

El cuerpo de un frame, a su vez, puede adoptar nueve formas diferentes (detalladas en figura 10); ellas pueden ser:

- **open**: negocia los parámetros de conexión entre el bróker y el cliente.
- **begin**: indica que una conexión se inicia.
- **attach**: se adjunta un enlace al mensaje, necesario para poder usar la transferencia de datos.
- **flow**: cambia el estado de un enlace.
- **transfer**: con el marco de la transferencia se transfiere el mensaje real.
- **disposition**: un frame de disposición informa sobre modificaciones en la entrega de la información.
- **detach**: elimina el enlace.
- **end**: indica que se termina la conexión.
- **close**: termina la conexión y declara que no se van a enviar más frames.

El tamaño del payload es negociable, sin límite[2], y no está definido; aunque las recomendaciones indican no enviar mensajes de más de 128MB. El peso del mensaje y la transferencia dependerá del tamaño del payload

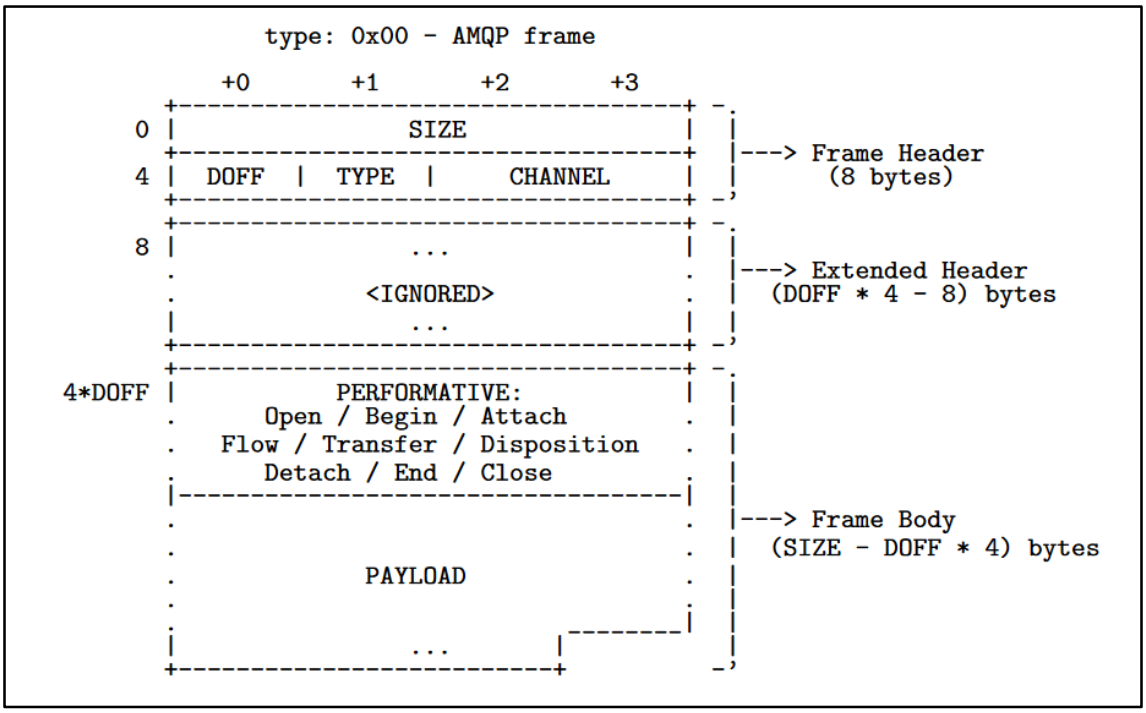


Figura 10-A. Estructura de una trama AMQP
Fuente: Estandar AMQP. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

[2] https://www.cloudamqp.com/blog/2019-05-24-what-is-the-message-size-limit-in-rabbitmq.html ; consultado el 20 de marzo de 2020

El PAYLAOD contiene un total de 7 áreas de datos: encabezado, anotaciones de entrega, anotaciones de mensaje, propiedades, propiedades de la aplicación, datos de la aplicación y pie de página. El papel de estos elementos es el siguiente (30):

- Encabezado: La sección de encabezado registra el estado de interacción del mensaje AMQP en el 'middleware habilitado para AMQP'. Por ejemplo, información como el número total de veces que este mensaje se ha intercambiado entre nodos, prioridad y valor de Tiempo de vida (TTL).

- Delivery-Annotations : Solo los atributos específicos del encabezado, estándar y definidos por ISO / IEC se pueden pasar en el encabezado. El área de datos de anotaciones de entregas se utiliza para registrar información de encabezado *'no estándar*'.

- Anotaciones de mensaje: se utiliza para almacenar algunas propiedades auxiliares personalizadas. A diferencia de la información no estándar en el área de anotaciones de entrega, los atributos personalizados aquí se utilizan principalmente para la conversión de mensajes. Por ejemplo, el proceso de conversión de información AMQP-JMS realizará JMS de acuerdo con los atributos "x-opt-jms-type", "x-opt-to-type", "x-opt-reply-type" y "name" de esta área de datos

- Propiedades: El atributo Propiedades registra atributos 'inmutables' del cuerpo del mensaje AMQP. En la sección de propiedades, solo se pueden pasar propiedades estandarizadas, estándar y definidas por ISO / IEC. Por ejemplo: identificación del mensaje, identificación del paquete, identificación del remitente, codificación de contenido, etc.

- Propiedades de la aplicación: propiedades de "datos de la aplicación", en las que los datos relacionados con la aplicación se registran; dado que necesitan usar esta parte de los datos para determinar su lógica de procesamiento. Por ejemplo: qué Exchange se envía, cuál es el valor de enrutamiento del mensaje, si persiste o no.

- Application-data: parte del del mensaje AMQP descrito en formato binario.

- Pie de página: generalmente almacena contenido auxiliar en esta área de datos, como hash de mensajes, HMAC, firma o detalles de cifrado.

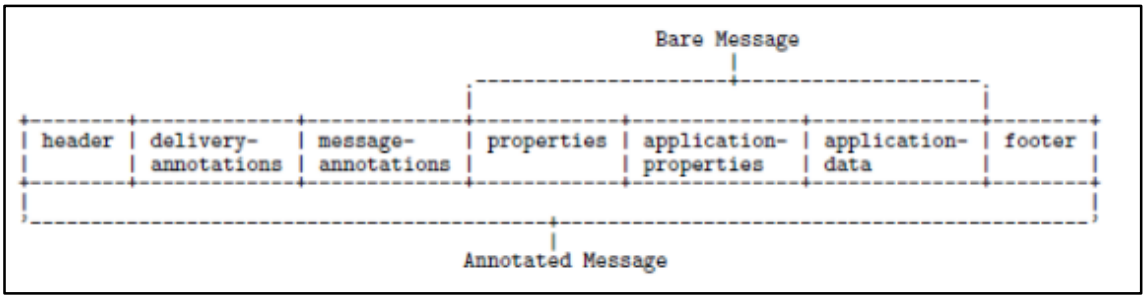


Figura 10-B. Payload en la estructura de una trama AMQP
Fuente: Estandar AMQP. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

Aspectos relativos a la Seguridad

AMQP provee una serie de mecanismos de seguridad que permiten fortalecer las conexiones entre clientes y servidores y asegurar que los datos transmitidos, almacenados y procesados estén adecuadamente protegidos.

Las capas de seguridad se utilizan para establecer un transporte autenticado y/o cifrado sobre el cual se puede canalizar el trafico AMQP. Las capas propuestas se pueden canalizar entre sí (una capa de seguridad utilizada por dos pares para autenticarse, podría canalizarse a través de una capa de seguridad establecida con fines de cifrado, por ejemplo) (31).

Como herramientas principales el protocolo permite la utilización de TLS para el cifrado del transporte y el uso de una capa extra de seguridad conocida como SASL *(capa de seguridad y autenticación simple - Simple Authentication and Security Layer)*.

Existen, fundamentalmente, varios mecanismos SAS admitidos y soportados para la autenticación. Muchos de ellos proveen autenticación ya encriptada como CRAM-MD5, GSSAPI o DIGEST-MD5 (32); sin embargo los más utilizado son ANONYMOUS y PLAIN (33). En el caso de ANONYMOUS no se pasa ninguna credencial del cliente al gestor de colas para la autenticación; si el gestor de colas tiene especificado como requerido un atributo de autenticación, rechazara la conexión. En el caso de PLAIN se pasa el nombre de usuario y la contraseña del cliente al gestor de colas para la autenticación. En este caso, si el gestor esta esperando un user/password y no se envía nada se rechaza la conexión; y si se envía se valida que sean los correctos. AMQP, dependiendo la implementación, permite validar estas credenciales contra un servidor externo como LDAP. Una vez validados las credenciales, también pueden validarse los permisos sobre el Exchange o la Cola; esto también es permitido por el protocolo.

Debe tenerse en cuenta que en modo PLAIN las credenciales de usuario viajan en texto plano y son factibles de ser interceptadas y de sufrir un ataque MIM (Man in the middle); por eso se sugiere cifrar la comunicación con TLS en caso de usar este mecanismo.

Como cualquier canal de comunicación, el protocolo AMQP es potencialmente vulnerables frente a ataques de suplantación, denegación de servicios, inyección y modificación

de datos, entre otros. Es recomendable que se utilicen todas las características de seguridad que el protocolo provee, como **la encriptación utilizando TLS/SSL en el transporte**. La longitud y fortaleza de las contraseñas vuelve a tener una gran relevancia en IoT: debido a pobres configuraciones de estas, podemos exponer muchos dispositivos vinculados.

4 - HTTP

HTTP es el protocolo más importante para Internet tal como se lo conoce hoy. Impulsa la World Wide Web (WWW) y es compatible con prácticamente cualquier plataforma y lenguaje de programación. HTTP se ha vuelto flexible a lo largo de los años, evolucionando en soluciones que inicialmente presentaba el protocolo, superando esas deficiencias, y convirtiéndolo hoy en una opción popular para la comunicación de IoT (26).

HTTP utiliza un modelo clásico de comunicación de solicitud / respuesta, que permite al cliente enviar datos al servidor y recibir una respuesta después que el procesamiento finalizó en el lado del servidor. Si bien este modelo es perfecto para el World Wide Web (un cliente puede solicitar una página web y el servidor entrega la página), la comunicación en IoT, impulsada por eventos fundamentalmente, no resulta del todo optima con HTTP.

Una ventaja de HTTP es la flexibilidad del protocolo. Esto se debe a la capacidad de agregar encabezados arbitrarios a cualquier solicitud y respuesta. Hay muchos encabezados estandarizados que son universalmente aceptados, como autenticación *(authentication)*, control de caché (cache control) y preferencias del cliente para la codificación y el idioma de la respuesta.

Características

Diseñado a principios de la década de 1990, HTTP es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación, y se transmite sobre el protocolo TCP, o el protocolo encriptado TLS, aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), sino que además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto. Algunas características claves del protocolo son: (34)

- **HTTP es sencillo**

  Incluso con el incremento de complejidad, que se produjo en el desarrollo de la versión del protocolo HTTP/2, en la que se encapsularon los mensajes, HTTP está pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más fácil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.

- **HTTP es extensible**

  Presentadas en la versión HTTP/1.0, las cabeceras de HTTP, han hecho que este protocolo sea fácil de ampliar y de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.

- **HTTP es un protocolo con sesiones, pero sin estados**

  HTTP es un protocolo sin estado, es decir: no guarda ningún dato entre dos peticiones en la misma sesión. Pero, mientras HTTP ciertamente es un protocolo sin estado, el uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. Usando la capacidad de ampliación del protocolo HTTP, las cookies permiten crear un contexto común para cada sesión de comunicación.

La característica del protocolo HTTP de ser ampliable, ha permitido que durante su desarrollo se hayan implementado más funciones de control y funcionalidad sobre la Web: caché o métodos de identificación o autentificación fueron temas que se abordaron pronto en su historia. Se presenta a continuación una lista con los elementos que se pueden controlar con el protocolo HTTP (34):

- ***Cache:*** El cómo se almacenan los documentos en la caché, puede ser especificado por HTTP. El servidor puede indicar a los proxies y clientes, que quiere almacenar y durante cuánto tiempo. Aunque el cliente, también puede indicar a los proxies de caché intermedios que ignoren el documento almacenado.

- ***Flexibilidad del requisito de origen:*** Para prevenir invasiones de la privacidad de los usuarios, los navegadores Web, solamente permiten a páginas del mismo origen, compartir la información o datos. Esto es una complicación para el servidor, así que mediante cabeceras HTTP, se puede flexibilizar o relajar esta división entre cliente y servidor

- ***Autentificación:*** Hay servicios web que pueden estar protegidos, de manera que solo los usuarios autorizados puedan acceder. HTTP provee de servicios básicos de autentificación, por ejemplo mediante el uso de cabeceras como: WWW-Authenticate, o estableciendo una sesión especifica mediante el uso de HTTP cookies.

- ***Proxies y tunneling:*** Servidores y/o clientes pueden estar en intranets y esconder así su verdadera dirección IP a otros. Las peticiones HTTP utilizan los proxies para acceder a ellos. Pero no todos los proxies son HTTP proxies. El

protocolo SOCKS, por ejemplo, opera a un nivel más bajo. Otros protocolos, como el FTP, pueden ser servidos mediante estos proxies.

- **Sesiones:** El uso de HTTP cookies permite relacionar peticiones con el estado del servidor. Esto define las sesiones, a pesar de que por definición el protocolo HTTP es un protocolo sin estado.

Arquitectura

HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error). De los protocolos más comunes en Internet, TCP es fiable. Por lo tanto **HTTP, se apoya en el uso del protocolo TCP que está orientado a conexión**, aunque una conexión continua no es necesaria siempre.

En la versión del protocolo HTTP/1.0, existía una conexión TCP por cada petición/respuesta intercambiada, presentando esto dos grandes inconvenientes: abrir y crear una conexión requiere varias idas y vueltas de mensajes y por lo tanto resultaba lento. Para atenuar estos inconvenientes, la versión del protocolo HTTP/1.1 presentó el *'pipelining'* y las conexiones persistentes: el protocolo TCP que lo transmitía en la capa inferior se podía controlar parcialmente, mediante la cabecera 'Connection'. La versión del protocolo **HTTP/2** fue más allá y **usa multiplexación de mensajes sobre un única conexión**, siendo así una comunicación más eficiente.

En la versión HTTP/2 **se permite la compresión del encabezado**; con HTTP/1.1, todos los encabezados deben transmitirse para cada solicitud. Los navegadores web a menudo envían varios encabezados que suman varios kilobytes. HTTP/2 permite comprimir los encabezados para reducir la sobrecarga. Otra funcionalidad incorporada en HTTP/2 reside en la habilitación de envío de datos a un cliente, por parte del servidor, de manera proactiva, si el recurso ha cambiado

Todas las mejoras descriptas han resultado en una importante disminución de la sobrecarga, lo que implica una baja en la latencia y transforma a **HTTP/2 en un buen candidato para un protocolo de solicitud/respuesta en el contexto de IoT**.

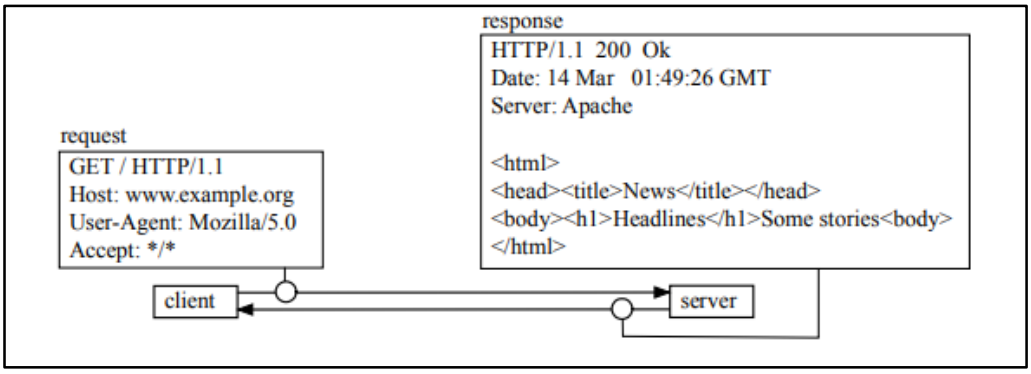


Figura 11. Patrón de solicitud / respuesta. El cliente inicia una solicitud y el servidor envía una respuesta tan pronto como procese la solicitud. La figura ilustra una solicitud de ejemplo y una respuesta de ejemplo para HTTP / 1.1
Fuente: The Technical Foundations of IoT. ISBN 9781630812515.

RESTful APIs

Un enfoque HTTP y centrado en la web para diseñar API web fue sugerido por Roy Fielding, uno de los principales autores de la especificación HTTP, en 2000 al introducir el término *representational state transfer* –transferencia de un estado representacional- o **REST**.

REST implica el uso de HTTP de forma explícita y de manera que sea coherente con la definición del protocolo. Este principio básico del diseño de REST establece una correlación individual entre las operaciones de crear, leer, actualizar y borrar (CRUD) y los métodos HTTP. Según esta correlación (35):

- Para crear un recurso en el servidor hay que utilizar un POST.
- Para recuperar un recurso hay que utilizar un GET.
- Para cambiar el estado de un recurso, o para actualizarlo, hay que utilizar un PUT.
- Para eliminar o borrar un recurso hay que utilizar un DELETE.

No existe un estándar único para API RESTful, pero diferentes codificaciones como JavaScript Object Notation (**JSON**) o Extensible Markup Language (**XML**) se utilizan en diferentes niveles conceptuales, junto con HTTP, identificadores uniformes de recursos (URI) y TLS.

Las API RESTful diferencian entre URI para colecciones y recursos. Un recurso es un punto final individual en particular con un único elemento lógico que normalmente se representa en JSON, XML, HTML o cualquier otro formato. Una colección es una acumulación de recursos individuales.

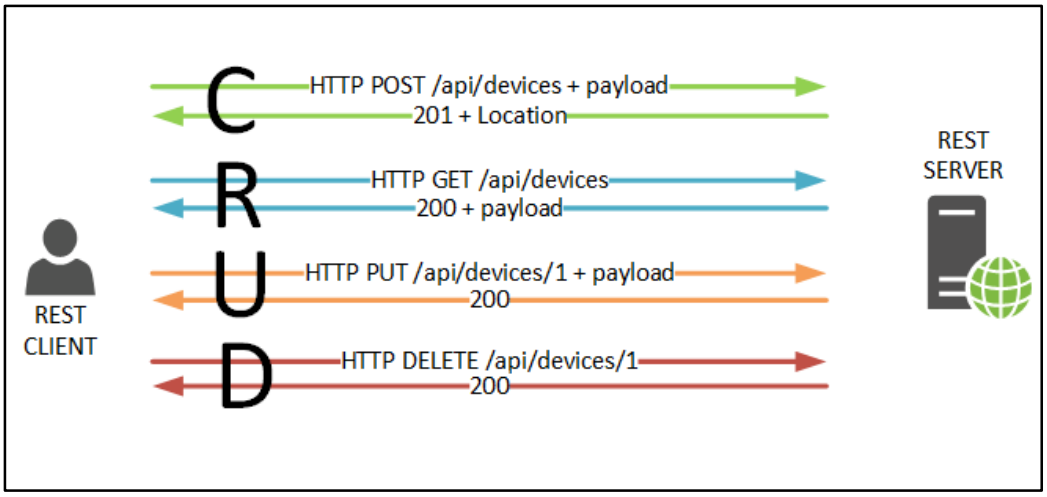


Figura 12. Envio de solicitudes / respuestas mediante HTTP / REST
Fuente: API REST. URL: https://jonmircha.com/api-rest

Según se trate de una petición a un recurso o a una colección, utilizando Restful APIs en HTTP, se puede categorizar y clasificar de la siguiente manera (26).

Método	En Recursos	En colecciones
GET	Devuelve el elemento o recurso	Devuelve las URIs de los miembros de la colección
POST	Generalmente no utilizado en recursos simples	Crea una nueva entrada en la colección y devuelve la URI del nuevo recurso creado
PUT	Reemplaza o crea el recurso si no existe	Reemplaza toda la colección por una nueva *(bulk operation)*
DELETE	Elimina el recurso	Elimina toda la colección *(bulk operation)*

Ejemplos

Para el uso de IoT, como se describió anteriormente, se utilizan URI (Uniform Resource Identifier). Esto permite identificar de manera específica un recurso. Una URL que representa una colección de usuarios, podría accederse mediante, por ejemplo, *http://my.api.com/users*

Una URL que representa a un usuario individual con el nombre abc, siguiendo el mismo estándar, podría accederse mediante por ejemplo, http://my.api.com/users/abc

Según la representación de Amazon[3], en ejemplos donde utiliza su core IoT en AWS mediante HTTP, los clientes y dispositivos pueden publicar sus mensajes usando HTTP haciendo POST a un cliente específico y a un tópico específico, según la url

https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1

Donde *IoT_data_endpoint* es el dispositivo al cual se quiere acceder y *url_encoded_topic_name* es el nombre completo del tema del mensaje que se envía.

Como puede observarse, muchas grandes empresas de tecnología como Amazon proveen soporte para la utilización de IoT mediante HTTP, brindando incluso guías de implementación (36) y mejores prácticas; por lo **que es un protocolo en plena utilización en el mercado de IoT** y no debe dejarse de lado en este análisis.

Trama HTTP

Los mensajes HTTP, son los medios por los cuales se intercambian datos entre servidores y clientes. Hay dos tipos de mensajes: *peticiones*, enviadas por el cliente al servidor, para pedir el inicio de una acción; y *respuestas*, que son la respuesta del servidor.

Los mensajes HTTP están compuestos de texto, codificado en ASCII, y pueden comprender múltiples líneas. En HTTP/1.1, y versiones previas del protocolo, estos mensajes eran enviados de forma abierta a través de la conexión. En HTTP/2.0 los mensajes, que anteriormente eran legibles directamente, se conforman mediante tramas binarias codificadas para aumentar la optimización y rendimiento de la transmisión (37).

Las peticiones y respuestas HTTP, comparten una estructura similar, compuesta de:

- Una línea de inicio ('start-line') describiendo la petición a ser implementada, o su estado, sea de éxito o fracaso. Esta línea de comienzo, es siempre una única línea.
- Un grupo opcional de cabeceras HTTP, indicando la petición o describiendo el cuerpo ('body') que se incluye en el mensaje.
- Una línea vacía ('empty-line') indicando toda la meta-información ha sido enviada.
- Un campo de cuerpo de mensaje opcional ('body') que lleva los datos asociados con la petición, o los archivos o documentos asociados a una respuesta. La presencia del cuerpo y su tamaño es indicada en la línea de inicio y las cabeceras HTTP.

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la *cabeza* de la peticiones, mientras que su contenido en datos se conoce como el *cuerpo* del mensaje.

[3] Implementando Iot con el core de AWS: https://docs.aws.amazon.com/iot/latest/developerguide/http.html. Consultado el 20 de mayo de 2021

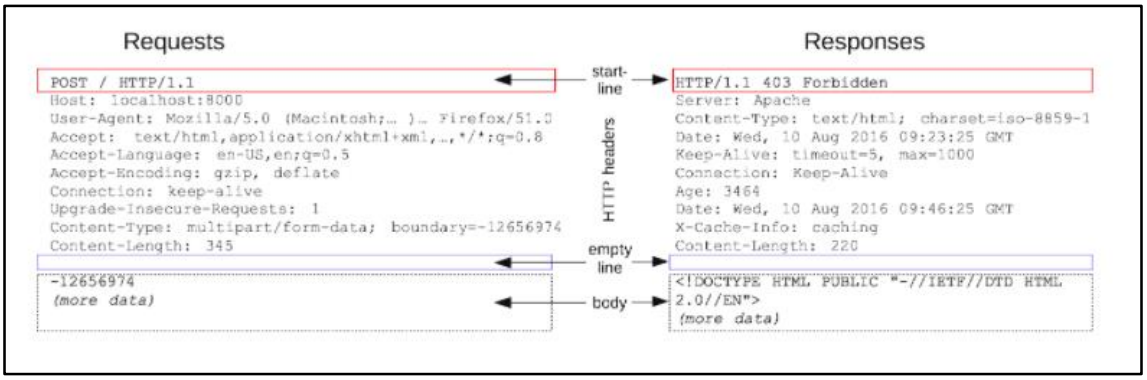


Figura 13. Mensaje HTTP para Requests y mensaje HTTP para Responses
Fuente: Mensajes HTTP- MDN Web Docs. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

Peticiones (Request)

Una petición HTTP está compuesta de una línea de inicio, una o varias cabeceras y un cuerpo de mensaje.

Línea de inicio

Las peticiones HTTP son mensajes enviados por un cliente, para iniciar una acción en el servidor. Su línea de inicio está formada por tres elementos:

- **Un método HTTP**, un verbo como: GET, PUT o POST o un nombre como: HEAD u OPTIONS; que describan la acción que se pide sea realizada.
- **El objetivo de la petición**, que en IOT normalmente es una URI indicando la dirección del recurso objetivo
- **La versión de HTTP**, que cual define la estructura de los mensajes, actuando como indicador, de la versión que espera que se use para la respuesta.

Un mensaje request hacia una uri solicitando los Gateway Wireless, por ejemplo, podría ser [4]:

GET */wireless-gateways?maxResults=MaxResults&nextToken=NextToken* **HTTP/1.1**

En el objetivo de la petición pueden incluirse parámetros, siguiendo la convención utilizada para el protocolo, donde (?) indica el inicio de parámetros y (&) la separación entre cada uno de ellos.

[4] AWS IoT Wireless API Reference: https://docs.aws.amazon.com/es_es/iot-wireless/2020-11-22/apireference/API_ListWirelessGateways.html; consultado el 20 de mayo de 2021

Cabeceras

Las cabeceras HTTP de una petición siguen la misma estructura que la de una cabecera HTTP. Una cadena de caracteres, que no diferencia mayúsculas ni minúsculas, seguida por dos puntos (':') y un valor cuya estructura depende de la cabecera. La cabecera completa, incluido el valor, ha de ser formada en una única línea, y pude ser bastante larga.

Hay bastantes cabeceras posibles. Estas se pueden clasificar en varios grupos:

- Cabeceras generales:('General headers'), afectan al mensaje como una unidad completa.
- Cabeceras de petición: ('Request headers'), como *User-Agent*, *Accept-Type*, modifican la petición especificándola en mayor detalle (como: Accept-Language (en-US), o dándole un contexto, como: *Referer*
- Cabeceras de entidad, ('Entity headers'), como *Content-Length* las cuales se aplican al cuerpo de la petición.

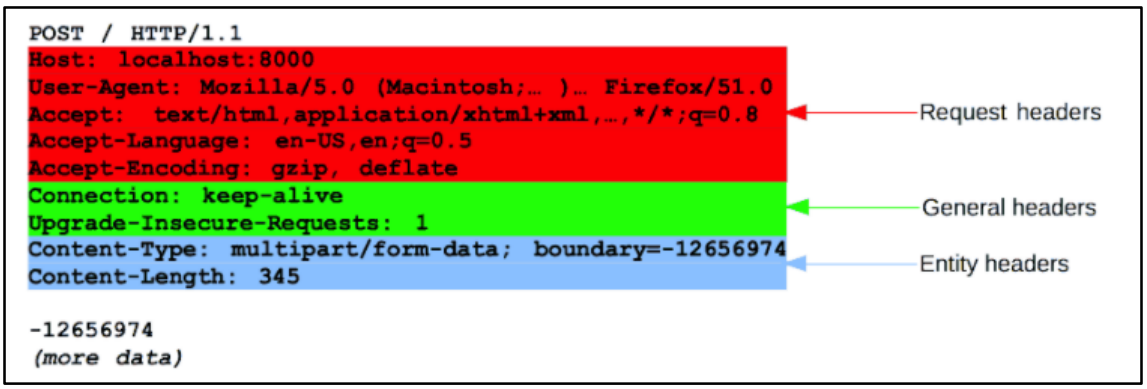


Figura 14. Detalle de una cabecera HTTP en solicitud de mensaje
Fuente: Mensajes HTTP- MDN Web Docs. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

Cuerpo de Mensaje

La parte final de la petición es el cuerpo. No todas las peticiones llevan uno: las peticiones que reclaman datos, como GET, HEAD, DELETE, u OPTIONS, normalmente, no necesitan ningún cuerpo. Algunas peticiones pueden mandar peticiones al servidor con el fin de actualizarlo: como es el caso con la petición POST

Respuestas (Response)

Las respuestas, como se muestra en la Figura 13, tiene la misma estructura que la solicitud pero con algunas diferencias conceptuales.

Línea de estado

La línea de inicio de una respuesta HTTP, se llama la línea de estado, y contienen la siguiente información:

- La **versión del protocolo**, normalmente HTTP/1.1.
- Un **código de estado**, indicando el éxito o fracaso de la petición. Códigos de estado muy comunes son: 200, 404, o 302
- Un **texto de estado**, que es una breve descripción, en texto, a modo informativo, de lo que significa el código de estado, con el fin de que una persona pueda interpretar el mensaje HTTP.

Una línea de estado típica es por ejemplo: HTTP/1.1 404 Not Found.

Cabecera:

Las cabeceras HTTP para respuestas siguen también la misma estructura como cualquier otra cabecera: una cadena de texto, que no diferencia entre mayúsculas y minúsculas, seguida por dos puntos (':') y un valor cuya estructura depende del tipo de cabecera. Toda la cabecera incluida su valor, se ha de expresar en una única línea.

Existen varias cabeceras posibles. Estas se pueden dividir en distintos grupos:

- Cabeceras generales, ('General headers'), que afectan al mensaje completo.
- Cabeceras de respuesta, ('Response headers'), como Vary, Accept-Ranges, que dan información adicional sobre el servidor, que no tiene espacio en la línea de estado.
- Cabeceras de entidad, ('Entity headers'), como Content-Length las cuales se aplican al cuerpo de la petición.

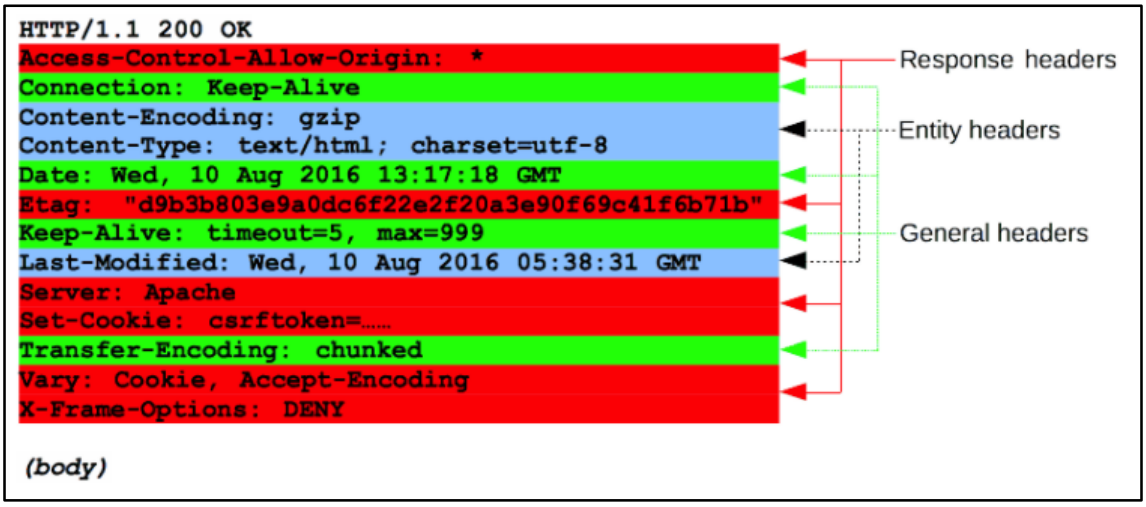


Figura 15. Detalle de una cabecera HTTP en una respuesta
Fuente: Mensajes HTTP- MDN Web Docs. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

Cuerpo:

La última parte del mensaje de respuesta es el 'body' o cuerpo. No todas las respuestas tienen uno, respuestas con un código de estado como 201 o 204 normalmente prescinden de él. De forma general, los cuerpos se pueden diferenciar en tres categorías:

- **Cuerpos con un único dato,** consisten en un simple archivo, de longitud conocida y definido en las cabeceras: Content-Type y Content-Length.
- **Cuerpos con un único dato con longitud desconocida,** consisten en un simple archivo, de longitud desconocida, y codificado en partes, indicadas con Transfer-Encoding valor *chunked*
- **Cuerpos con múltiples datos**, consisten de varios datos, cada uno con una sección distinta de información.

En IoT, muchas veces las respuestas entregadas a solicitudes realizadas a URI específicas son JSON o XML. Para el caso del ejemplo detallado con anterioridad (petición mediante GET de los *Wireless Gateway*) la respuesta podría ser:

```
HTTP/1.1 200
Content-type: application/json

{
   "NextToken": "string",
   "WirelessGatewayList": [
      {
         "Arn": "string",
         "Description": "string",
         "Id": "string",
         "LastUplinkReceivedAt": "string",
         "LoRaWAN": {
            "GatewayEui": "string",
            "JoinEuiFilters": [
               [ "string" ]
            ],
            "NetIdFilters": [ "string" ],
            "RfRegion": "string",
            "SubBands": [ number ]
         },
         "Name": "string"
      }
   ]
}
```

Figura 16. Ejemplo de una respuesta HTTP completa
Fuente: Amazon AWS IoT. URL: https://docs.aws.amazon.com/es_es/iot-wireless/2020-11-22/apireference/API_ListWirelessGateways.html

Diferencias con HTTP/2

Los mensajes HTTP/1.x tienen algunas desventajas por su no muy alta eficiencia en la transmisión.

- Las cabeceras, al contrario de los cuerpos, no se comprimen.
- Las cabeceras, que habitualmente se repiten de un mensaje al siguiente, aun así, deben ser enviadas siempre en todos los mensajes.
- No se puede multiplexar. Se han de abrir varias conexiones para el mismo servidor, las conexiones TCP 'en caliente' son más eficientes que las conexiones 'en frio'.

HTTP/2 introduce un paso extra: divide los mensajes HTTP/1.x en tramas que integra en un flujo de datos. Los datos y las tramas de las cabeceras, se separan, esto permite la compresión de las cabeceras. Varios flujos de datos pueden combinarse juntos, y entonces se puede usar un procedimiento de multiplexación, permitiendo un uso más eficiente, de las conexiones TCP.

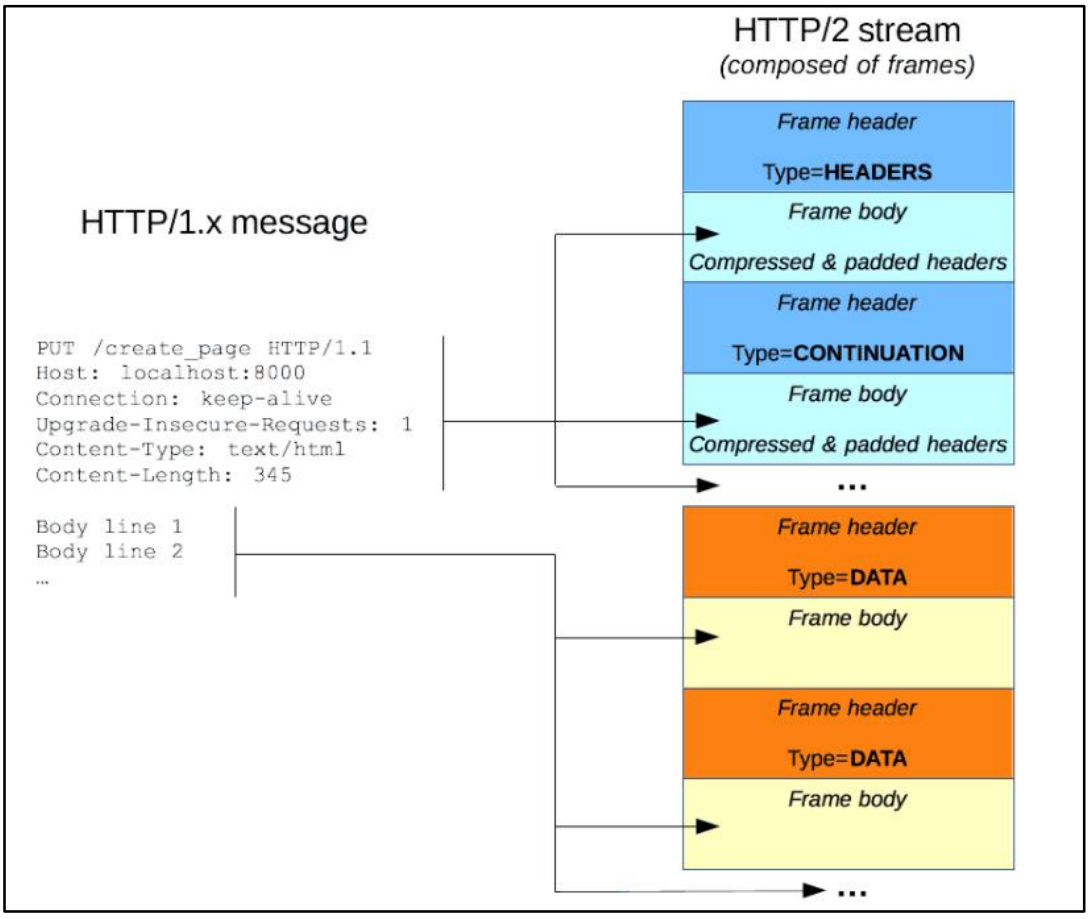


Figura 17. Diferencias entre tramas HTTP/1.X y HTTP/2
Fuente: Mensajes HTTP- MDN Web Docs. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

Aspectos relativos a la Seguridad

HTTP utiliza TLS *(Transport Layer Security)* para asegurar la comunicación entre el cliente y el servidor a nivel de transporte. Este nivel de seguridad implica cifrado de la comunicación de manera independiente al protocolo superior. Cuando TLS se encuentra correctamente implementado cualquier observador externo solo puede inferir los puntos finales de conexión, el tipo de cifrado y datos relativos a la frecuencia o cantidad de datos enviados; pero no puede leer ni modificar ninguno de los datos reales

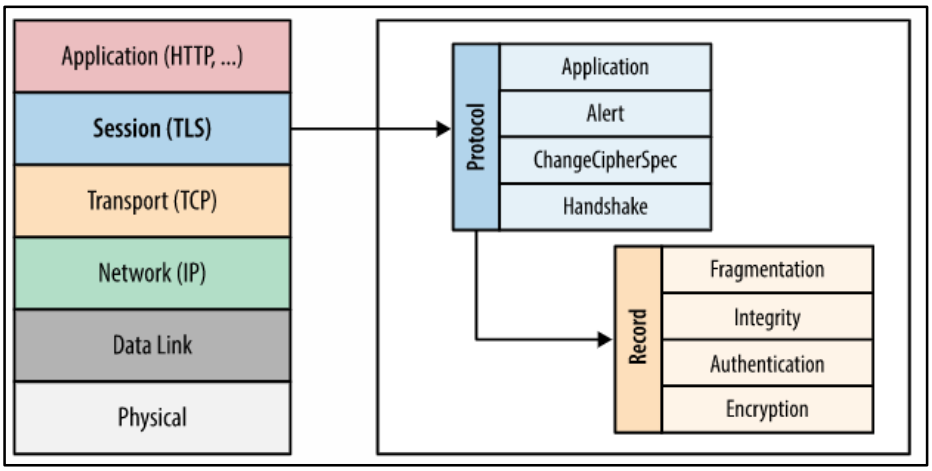


Figura 18. TLS en HTTP
Fuente: Transport Layer Security (TLS). URL: https://hpbn.co/transport-layer-security-tls/

Este cifrado, independiente de HTTP, busca proveer tres servicios esenciales a todas las aplicaciones que corren sobre el: encriptación, autenticación e integridad de datos. Técnicamente no es necesario usar los tres servicios en todas las transacciones, es decir, se puede aceptar un certificado sin validar su autenticidad por ejemplo, pero existen riesgos de seguridad altos al no hacerlo. En la práctica es importante aprovechar el uso de los tres servicios brindados por el protocolo

Para establecer un canal de envío de datos criptográficamente seguro, los pares de conexión (clientes y servidor) deberán acordar que conjuntos de cifrados se utilizan (*ciperhersuite*), la versión de TLS, y verificar certificados de ser necesario. El protocolo TLS especifica un *Handshake* para este intercambio. Desafortunadamente, cada uno de estos pasos requiere transferencias de paquetes entre el cliente y el servidor, lo que **agrega latencia de inicio a todas las conexiones TLS** (38) como muestra la figura 19

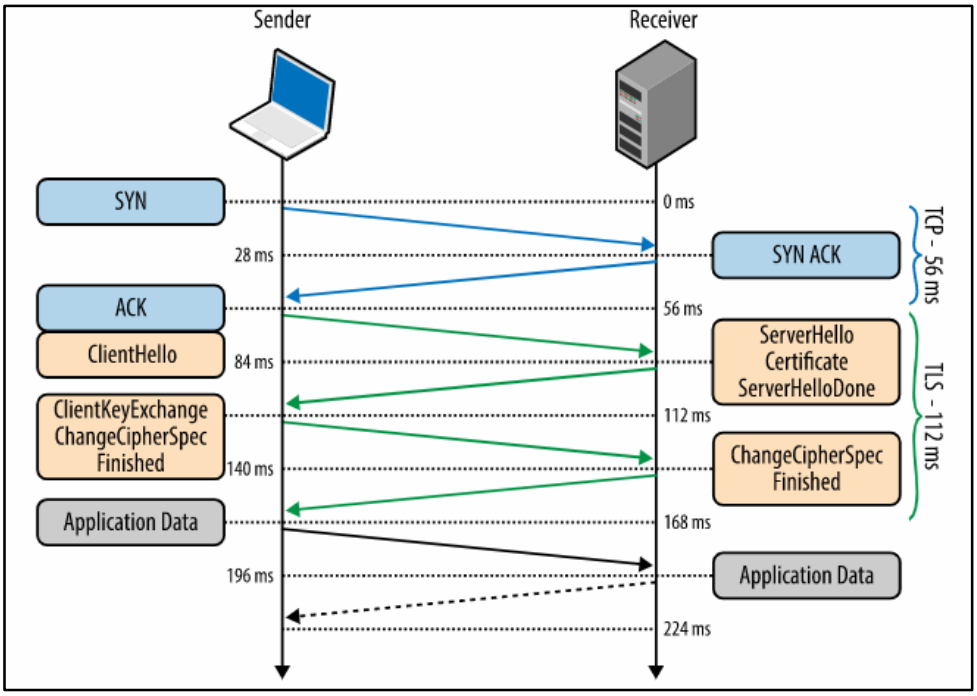


Figura 19. Handshake TLS
Fuente: Transport Layer Security (TLS). URL: https://hpbn.co/transport-layer-security-tls/

Autenticación HTTP

Además de TLS, utilizado para cifrar la comunicación en la capa de transporte, dos mecanismos de autenticación son especificados por HTTP, ambos usan el encabezado estandarizado *Authorization* (26).

- Autenticación Básica *(Basic Authentication)*: Es el mecanismo de autenticación más común; donde se envía el nombre de usuario y contraseña en la forma *user:password* codificado en base64. Este método de autenticación requiere TLS, de otra manera cualquier atacante podría capturar y leer el usuario y password en texto plano

- Autenticación de acceso con resumen (*Digest Access Authentication*): Este método requiere un *nonce* (cadena generada aleatoriamente) que se envía desde el servidor al cliente. El *nonce* se usará por el cliente para generar el hash que será enviado al servidor. El cliente calcula un hash MD5 de nombre de usuario, contraseña y otra información (incluida la cadena) y envía el valor *hasheado* al servidor. El servidor puede reconstruir el valor hasheado y determinar si la combinación usuario/contraseña utilizada para calcular el hash fue la correcta. La información de nombre de usuario y password nunca se envidan al servidor sin cifrar. De todas maneras, si ese método se utiliza sin cifrado de transporte como TLS, los ataques *man-in-the-middle* todavía son factibles.

5 - CoAP

Constrained Application Protocol (CoAP) es un protocolo de transferencia web diseñado especialmente para ser utilizado con nodos y redes restringidas en Internet de las cosas. El protocolo está diseñado para aplicaciones de máquina a máquina (M2M), como energía inteligente y automatización de edificios, según detalla la misma página oficial de CoAP (39).

CoAP es un estándar IETF abierto (RFC 7252)[5] y admite diferentes extensiones de protocolo, que también son reconocidas por el IETF. Se basa en el modelo de comunicación de cliente/servidor similar a HTTP, pero se emplea mayormente en interacciones M2M. CoAP, además, admite y soporta funciones adicionales que son especialmente útiles en escenarios de IoT.

CoAP proporciona un modelo de interacción solicitud / respuesta entre puntos finales, admite el descubrimiento integrado de servicios y recursos e incluye conceptos clave de la Web como URI (Universal Resource Identifier). El protocolo está diseñado para interactuar fácilmente con HTTP para la integración con la Web cumpliendo requisitos especializados como soporte de multidifusión, muy poca sobrecarga y simplicidad para entornos restringidos.

Características

Una de las principales características establecidas en el diseño del protocolo es el poco consumo de recursos, tanto en dispositivos como en redes de comunicación. Utilizando este principio, CoAP posee muchas características que lo hacen útil para el uso de aplicaciones y plataformas IoT. Algunas de ellas son (26) (39; 40):

- Ligero: CoAP tiene un encabezado fijo de 4 bytes para todos sus mensajes. El encabezado puede ir seguido de opciones binarias compactas y un *payload*, que es el mensaje de aplicación. Las solicitudes y las respuestas comparten el concepto de encabezado. CoAP usa UDP que es más ligero que TCP

- Integración con HTTP: CoAP usa semántica RESTfull, similar a HTTP, y redefine las acciones de GET, PUT, POST y DELETE. La similitud de HTTP y CoAP en su diseño RESTful permite la conversión de CoAP a HTTP a través de aplicaciones proxy entre ambos protocolos de manera transparente.

- Mensajes asincrónicos: Dado que los mensajes son transportados utilizando UDP, pueden ser enviados y recibidos *out-of-order* (fuera de línea); procesándose las

[5] https://tools.ietf.org/html/rfc7252; RFC7252: The Constrained Application Protocol (CoAP) . Consultado 15 diciembre de 2020

solicitudes y respuestas de manera asíncrona. Esta característica facilita la implementación tanto en clientes como servidores con alta demanda de envío o recepción de mensajes.

- Seguridad con DTLS: para establecer un canal de comunicación seguro y cifrado entre el cliente y servidor, CoAP aprovecha el protocolo *Datagram Transport Layer Security* (DTLS); que consiste en un protocolo de seguridad alternativo para UDP basado en TLS. Tiene un conjunto de características similares que se ocupan del reordenamiento de paquetes, pérdida de datagramas y grandes transferencias de datos. Se detallara más sobre esta característica en el apartado *Seguridad*.

- Built-in-discovery: CoAP tiene desarrollado un *descubrimiento integrado de recursos*. Esto permite a los clientes detectar que tipo de servicios tiene disponible o brinda un servidor CoAP en tiempo real. Este mecanismo es especialmente útil en escenarios de IoT, donde hay interacciones M2M de manera automatizada.

- Elección del modelo de datos: Como HTTP, CoAP puede transportar diferentes tipos de payloads y puede identificar qué tipo de carga útil se está utilizando. CoAP se integra con XML, JSON, CBOR, entre otros.

- Recursos observables: CoAP define una extensión del protocolo, mediante la RFC 7641[6], que permite observar recursos. Después de que el cliente envía una solicitud de observación y recibe el permiso correspondiente, todas las actualizaciones del recurso se realizan automáticamente (son enviadas desde el servidor al cliente). Como resultado, el cliente no necesita indagar sobre el recurso para actualizaciones. Los recursos observables son un mecanismo eficiente si los recursos se actualizan con frecuencia

Arquitectura

El protocolo se encuentra diseñado para trabajar sobre interacciones o intercambios de mensajes de forma asíncrona entre nodos por medio del protocolo de transporte UDP con una baja sobrecarga de cabeceras para reducir la complejidad al momento de analizar un mensaje. Un nodo, actuando como cliente, envía una o más peticiones sobre uno o más recursos alojados en un determinado servidor que atenderá la petición. El servidor responderá a la petición indicando si la petición recibida es exitosa o no.

El protocolo CoAP proporciona dos tipos de modelo de comunicación: **un modelo basado en solicitud/respuesta similar a HTTP**, y **un modelo basado en**

[6] https://tools.ietf.org/html/rfc7641 ; RFC7641: Observing Resources in the Constrained Application Protocol (CoAP). Consultado 15 diciembre de 2020

publicación/suscripción, denominado observador y definido mediante RFC 7641, como se explicó anteriormente.

Arquitectura de COAP siguiendo el modelo cliente / servidor

Basado en lo definido por el RFC7452 (40) el modelo de interacción de CoAP es similar al modelo cliente / servidor de HTTP. Sin embargo, las interacciones de M2M suelen resultar en una implementación de CoAP actuando tanto en roles de cliente como de servidor. La solicitud CoAP es equivalente a la de HTTP y es enviada por un cliente a solicitar una acción (usando un código de método) en un recurso (identificado por un URI) en un servidor. Luego, el servidor envía una respuesta con una respuesta código; esta respuesta puede incluir una representación de recursos.

A diferencia de HTTP, CoAP se ocupa de estos intercambios d**e forma asincrónica a través de un transporte orientado a datagramas como UDP**. Esto se hace usando una capa de mensajes que admita confiabilidad opcional (con retroceso exponencial). CoAP define cuatro tipos de mensajes: Confirmable, No confirmable, Reconocimiento, Reinicio.

Las solicitudes son enviadas mediante mensajes de tipo CON o NON, en los cuales, la petición ejecuta un método (GET, PUT, POST y DELETE) sobre un recurso, el cual viene identificado por una ruta contenida en el campo URI-Path del mensaje CoAP. Las respuestas son enviadas dependiendo del tipo de mensaje en el cual se envió el método de solicitud. Si el mensaje es de tipo CON, se debería generar un mensaje de acuse de recibo conteniendo una de las siguientes clases de código de respuesta. Los valores 'xx' se reemplazan con los códigos reales que identifican con mayor precisión el código de respuesta según la especificación del protocolo (41):

- **2.xx: (Success):** La petición fue recibida, entendida y aceptada correctamente por el servidor. Si el recurso fue recuperado con éxito, actualizado, creado o eliminado, entonces debería producir una respuesta exitosa.
- **4.xx: (Client Error):** La petición contiene una sintaxis errónea o no puede ser correctamente tratada por el servidor, por ejemplo: URI incorrectos, solicitudes de recursos inexistentes o recursos a los cuales el servidor no tiene acceso.
- **5.xx: (Server Error):** El servidor no puede tratar una petición aparentemente correcta.

Para el acuse de recibo de los mensajes por parte del servidor, se puede utilizar dos tipos de técnicas para la respuesta: “Piggy-backing” o “Separate”.

- **Piggy-backing:** En esta técnica la respuesta de tipo Piggy-backing se da cuando el servidor responde inmediatamente a una solicitud recibida de tipo CON y envía un mensaje de tipo ACK. Este tipo de respuestas se dan independientemente de si la respuesta por parte del servidor indica éxito o fallo al tratar la solicitud

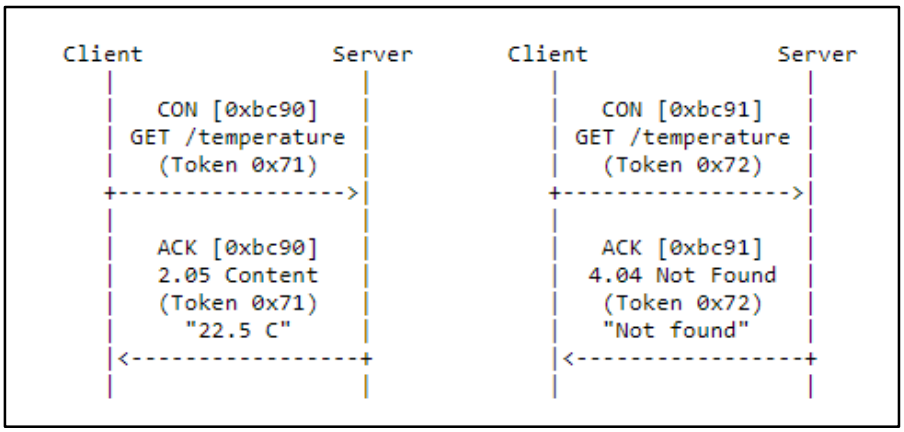


Figura 20. *Dos peticiones GET con respuestas Piggybacked*
Fuente: RFC7252. https://datatracker.ietf.org/doc/html/rfc7252

- **Separate:** Esta técnica es utilizada cuando al recibirse una petición en el servidor y este no sea capaz de responder de manera inmediata, ya sea porque no dispone temporalmente de acceso al recurso o porque se encuentra saturado. Si la petición se recibe a través de un mensaje CON y el servidor no puede enviar la respuesta de forma inmediata responde con un ACK confirmando que ha recibido la petición y que ésta será tratada tan pronto como sea posible. Posteriormente, la respuesta con el contenido del recurso se envía con un mensaje de tipo CON que deberá ser confirmado por el cliente para asegurar que éste último ha recibido correctamente la respuesta.

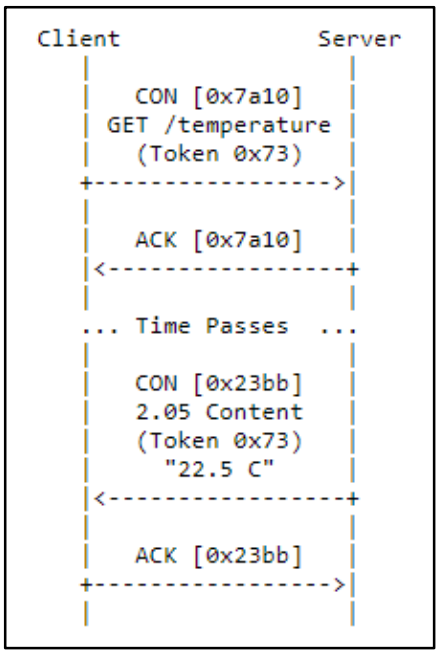


Figura 21.
Una peticion GET con respuesta Separate
Fuente: RFC - 7252
https://datatracker.ietf.org/doc/html/rfc7252

Dentro la comunicación con CoAP, un nodo puede tener más de un mensaje pendiente por confirmarse su entrega o simplemente el nodo puede interactuar con diferentes nodos a la

vez, por lo que se requiere un método para poder determinar que un mensaje recibido es la respuesta a un mensaje concreto y no a otro. Esto se consigue mediante el uso de un **Token**. El Token es un valor pensado para ser gestionado de forma local, para que un nodo cliente pueda diferenciar de forma concurrente las peticiones que tiene en curso. Cuando un nodo recibe un paquete que contiene un Token debe responder siempre manteniendo el mismo valor del Token; como puede observarse en la figura 20 y en la figura 21 (41). **Esto aplica para todas las tramas de CoAP**

Arquitectura de COAP siguiendo el modelo observador

Considerando que COAP ha sido definido como un protocolo para redes y nodos restringidos, y teniendo en cuenta que el estado de los recursos alojados en los servidores puede cambiar (inclusive de manera frecuente); la RFC 7461 (42) define una extensión del protocolo donde se enmarca el modelo observador.

Este modelo permite "*observar*" recursos, es decir, recuperar una representación de un recurso y mantener esta representación actualizada por el servidor durante un período de tiempo. El protocolo sigue un enfoque de mejor esfuerzo para enviar nuevas representaciones a los clientes y proporciona coherencia eventual[7] entre el estado observado por cada cliente y el estado real del recurso en el servidor.

Una de las características principales del modelo observador es permitir que un cliente pueda recibir continuamente respuestas de un servidor apoyándose en el descubrimiento de recursos. Es decir, el protocolo permite al cliente solicitar información de su interés en los recursos que tiene el servidor.

El modelo del observador permite que un nodo servidor envíe notificaciones continuamente hacia un cliente, después de que este se haya suscrito como observador mediante un mensaje de registro. El servidor mantiene una lista de todos los observadores registrados. El objetivo del servidor es mantener a los observadores actualizados sobre el evento observado, el cual puede ser enviado hacia el cliente a intervalos de tiempo regulares o cuando se produce un cambio en el valor del evento que se está observando (41). El modelo se basa en el patrón de diseño del observador, que es bien conocido en disciplina de ingeniería de software

El patrón de diseño *observador* se realiza en CoAP de la siguiente manera (42):

- **Asunto**: En el contexto de CoAP, el asunto es un recurso en un servidor CoAP. El estado del recurso puede cambiar a lo largo del tiempo, desde actualizaciones poco frecuentes hasta transformaciones y actualizaciones continuas.

[7] La coherencia eventual es una garantía teórica de que, mientras no se realicen actualizaciones nuevas en una entidad, todas sus lecturas mostrarán el último valor actualizado finalmente.

- **Observador**: un observador es un cliente de CoAP que está interesado en tener una representación actual del recurso en un momento dado.
- **Registro**: un cliente registra su interés en un recurso al iniciar una solicitud GET extendida al servidor. Además de devolver una representación del recurso de destino, esta solicitud hace que el servidor agregue el cliente a la lista de observadores del recurso.
- **Notificación**: siempre que cambia el estado de un recurso, el servidor notifica a cada cliente en la lista de observadores del recurso. Cada notificación es una respuesta CoAP adicional enviada por el servidor en respuesta a la solicitud GET extendida inicialmente recibida por el observador al realizar su registro e incluye una representación completa y actualizada del nuevo estado del recurso.

La Figura 22 muestra un ejemplo de un cliente CoAP que registra su interés en un recurso y recibir tres notificaciones: la primera con el estado actual en el momento del registro, y luego dos en los cambios al estado del recurso. Tanto la solicitud de registro como las notificaciones se identifican como tales por la presencia del ***Observe***. Todas las notificaciones llevan el token especificado por el cliente, por lo que el cliente puede correlacionarlos fácilmente con la solicitud.

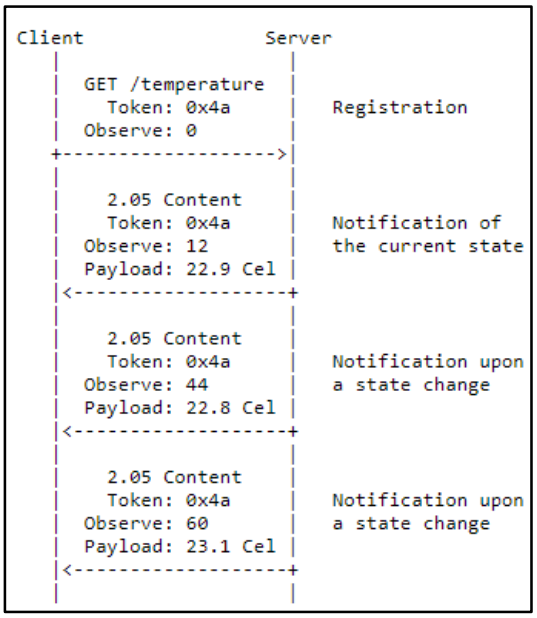


Figura 22.
Observando un recurso en CoAP
Fuente: RFC – 7641
https://datatracker.ietf.org/doc/html/rfc7641

Un cliente permanece en la lista de observadores mientras el servidor pueda determinar el interés continuo del cliente en el recurso. El servidor puede enviar una notificación en un mensaje *CoAP confirmable* solicitando un acuse de recibo del cliente. Cuando el cliente da de baja, rechaza una notificación o la transmisión de una notificación se agota después de varios intentos de transmisión, se considera que el cliente ya no está interesado en el recurso y es eliminado por el servidor de la lista de observadores.

Consistencia del modelo

Mientras un cliente está en la lista de observadores de un recurso, el objetivo del protocolo es mantener el estado del recurso observado por el cliente lo más sincronizado posible con el estado real en el servidor. No se puede evitar que el cliente y el servidor se queden fuera de sincronizar a veces; recordando que trabajan de manera offline mediante UDP. Además de que pueden existir varias causales:

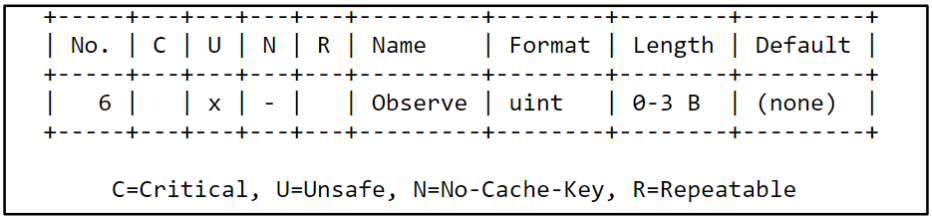
- siempre hay algo de latencia entre el cambio del estado del recurso y la recepción de la notificación.
- los mensajes CoAP con notificaciones pueden perderse, lo que provocará que el cliente asuma un estado anterior hasta que reciba una nueva notificación
- el servidor puede llegar a la conclusión errónea de que el cliente ya no está interesado en el recurso, lo que provocará que deje de enviar notificaciones y el cliente deberá asumir un estado anterior del recurso hasta que finalmente registre su interés nuevamente

El protocolo aborda este problema de la siguiente manera (42):

- Sigue un enfoque de *mejor esfuerzo* para enviar el estado actual del recurso después de un cambio de estado: los clientes deben ver el nuevo estado después de un cambio de estado tan pronto como sea posible, y deberían ver tantos estados como sea posible
- Etiqueta las notificaciones con una *duración máxima aceptable* para que el estado observado y el estado real estén fuera de sincronización. Cuando el tiempo de la notificación recibida alcance este límite, el cliente no puede utilizar la representación enviada hasta que reciba una nueva notificación o actualización.
- Está diseñado sobre el principio de consistencia eventual: el protocolo garantiza que si el recurso no se somete a un nuevo cambio de estado, todos los observadores registrados tendrán una representación actual del estado más reciente del recurso.

Aplicación del modo observador

La opción de modo observador es transparente y no cambia la forma o modos de comunicación que sigue el modelo de solicitud/respuesta; dado que como se explicó se trata de una respuesta continua ante un cambio del valor observado. El modelo sigue las siguientes propiedades



No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figura 23.
Tabla de opciones del modo observador
Fuente: RFC – 7641. https://datatracker.ietf.org/doc/html/rfc7641

Cuando se incluye en una solicitud GET, el modo observador extiende para que además de obtener el estado actual del recurso, también solicite al servidor agregarse o eliminarse en la lista de observadores del recurso en función del valor de la opción. La lista consiste en el número de dispositivo (endpoint cliente) y el token especificado por el cliente en la solicitud. Los posibles valores son: 0 (registro) agrega la entrada a la lista, si no está presente; 1 (cancelar registro) elimina la entrada de la lista, si está presente.

La opción de observación no es crítica para procesar la solicitud. Si el servidor está imposibilitado de agregar una nueva entrada a la lista de observadores, la solicitud vuelve a ser una solicitud GET normal y la respuesta no incluye la opción de observación (42).

La figura 24 ejemplifica como un cliente se *suscribe* a las notificaciones utilizando *Observe en 0*, para finalmente indicarle al servidor que no desea recibir más notificaciones, utilizando *Observe en 1*

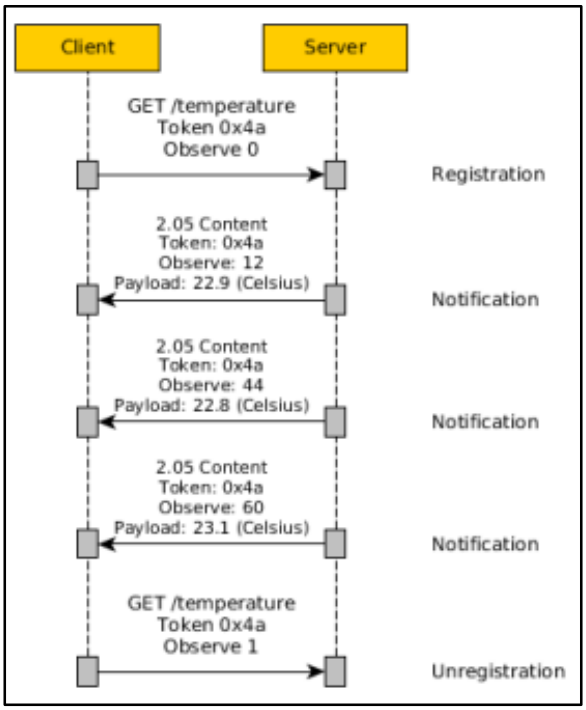


Figura 24.
Solicitud de observer, para luego desubscribirse
Fuente: RFC – 7641.
https://datatracker.ietf.org/doc/html/rfc7641

Tramas en CoAP

Como se ha definido, CoAP se basa en el intercambio de mensajes compactos que, por defecto, se transportan a través de UDP. Los mensajes están codificados en formato binario simple; el mensaje comienza con un encabezado corto de longitud fija (4 bytes) que puede ser seguido de opciones binarias compactas (tokens) entre 0 y 8 bytes de longitud.

Continuo a los tokens se encuentra una secuencia de cero o más opciones de CoAP en formato Tipo-Longitud-Valor (*Type-Lenght-Value TLV*), opcionalmente seguidas de una carga útil o *payload* que ocupa el resto del datagrama.

Los campos del encabezado, según la Figura 25, se definen de la siguiente manera:

- **Versión (Ver):** entero sin signo de 2 bits. Indica la versión de CoAP
- **Tipo (T):** entero sin signo de 2 bits. Indica si este mensaje es de tipo Confirmable (0), No confirmable (1), Reconocimiento (2) o Restablecer / Reinicio (3).
- **Longitud del token (TKL):** entero sin signo de 4 bits. Indica la longitud del campo Token de longitud variable (0-8 bytes).
- **Código (Code)**: entero sin signo de 8 bits, dividido en una clase de 3 bits (bits mas significativos) y un detalle de 5 bits (bits menos significativos). La clase puede indicar una solicitud (0), una respuesta exitosa (2), una error en la respuesta el cliente (4) o un error en la respuesta del servidor (5).
- **ID de mensaje (Message ID)**: entero sin signo de 16 bits. Usado para detectar mensajes duplicados y hacer coincidir mensajes de tipo reconocimiento/restablecimiento con mensajes de tipo Confirmable / No Confirmable

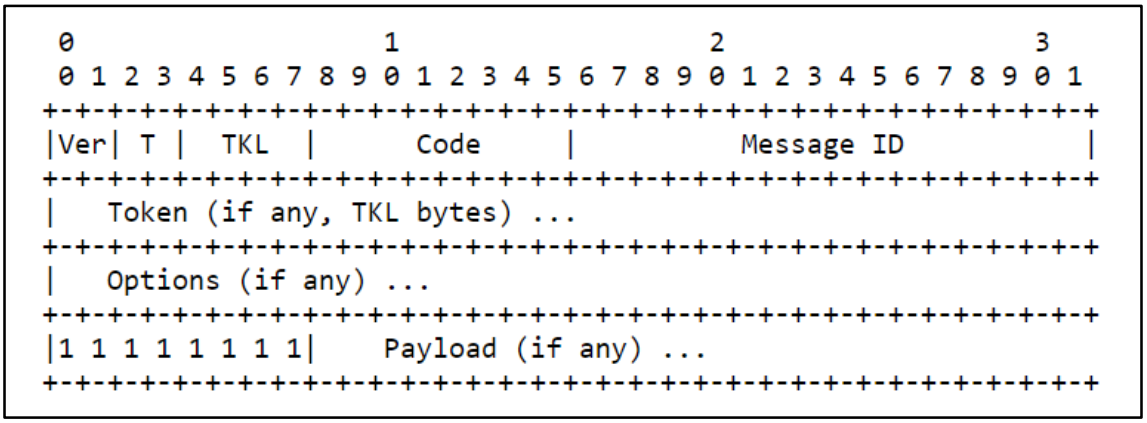


Figura 25- A.
Formato de un mensaje CoAP
Fuente: RFC – 7252. https://datatracker.ietf.org/doc/html/rfc7252

El encabezado va seguido del valor del token, que puede ser de 0 a 8 bytes, como indica en el campo Longitud del token. El valor del token se usa para correlacionar solicitudes y respuestas.

El encabezado y el token van seguido (si hubiere) de las opciones. Una opción puede ir seguida del final del mensaje (mensaje sin payload o carga útil), de otra opción (mensaje con múltiples opciones), o por el marcador de carga útil y la carga útil (*payload*).

Siguiendo el encabezado, el token y las opciones, si las hay, viene opcionalmente la carga útil. Si está presente y tiene una longitud distinta de cero, esta precedido de un prefijo de

un solo byte, que indica el final de opciones y el inicio de la carga útil. Los datos de carga útil se extienden desde el marcador hasta el final del datagrama UDP

Opciones en la trama

CoAP define una serie de opciones que pueden ser incluidas en un mensaje, especificadas por el "número de opciones" definido, su longitud y valor. Estas instancias (opciones de mensajes) son calculadas por la suma de los números de opciones del mensaje anterior, que deben aparecer en el orden definido más un delta de codificación.

El formato de las opciones es el siguiente:

- Opción Delta: entero son signo de 4 bits que indica la diferencia entre el número de opción actual del anterior.
- Longitud (length): entero sin signo de 4 bits que como su nombre indica la longitud del valor de la opción, si este campo se establece a 15, se añade un entero sin signo de 8 bits que permite longitudes que van desde 15 hasta 270 bytes, como se muestra en la figura 25-B.

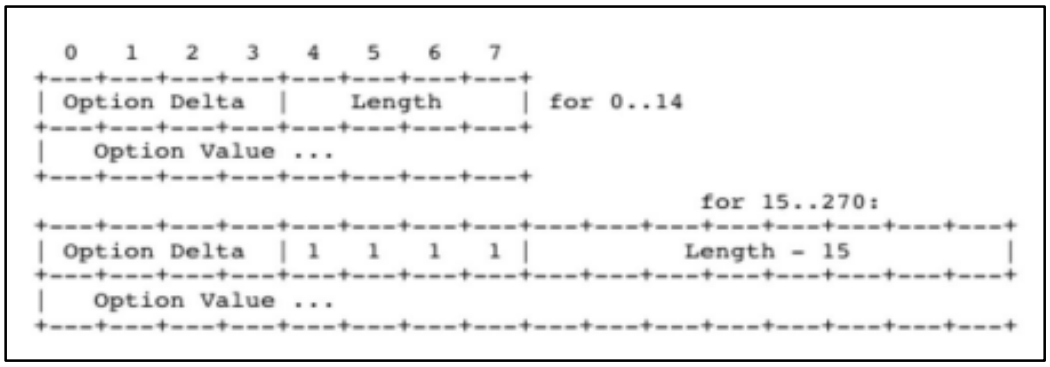


Figura 25- B.
Formato de opciones un mensaje CoAP
Fuente: RFC – 7252. https://datatracker.ietf.org/doc/html/rfc7252

Existen varias opciones CoAP, pero las más utilizadas son (43):

- Token: sirve para asociar peticiones con sus respuestas asociadas
- Uri-Host: se especifica para solicitar el host de Internet del recurso.
- Uri-Port: número de puerto del recurso.
- Uri-Path: segmento de la ruta absoluta al recurso.
- Uri-Query: cadena de consulta.
- Proxy-Uri: se utiliza para hacer una petición a un proxy.
- Content-Type: se indica el formato de representación de la carga útil (payload) del mensaje dado como un valor numérico.

- Max-Age: tiempo de respuesta en caché que el cliente tiene para esperar la respuesta del servidor.
- E-Tag: una respuesta proporciona el valor actual de la entidad-tag para la representación del recurso de destino. Una entidad-Tag es como un identificador local de recursos para diferenciar las representaciones de recursos que varían con el tiempo.
- Location-Path and Location-Query: indica la ubicación de un recurso, creado por un POST, como la ruta URI absoluta.
- If-Match: se utilizar para hacer una solicitud condicional de una actual existente o de un valor E-tag para una o más representaciones del recurso destino.
- If-None-Match: útil para solicitudes de creación de recursos, tales como peticiones PUT y como un medio de protección que evita sobrescribir IDs de clientes cuando varios de estos están ejecutándose en paralelo en el mismo recurso.

Aspectos relativos a la Seguridad

El estándar CoAP no especifica ningún mecanismo de autenticación. La implementación de CoAP de manera segura se basa en la correcta configuración y ejecución del transporte seguro de sus datagramas, utilizando *Datagram Transport Layer Security* (DTLS).

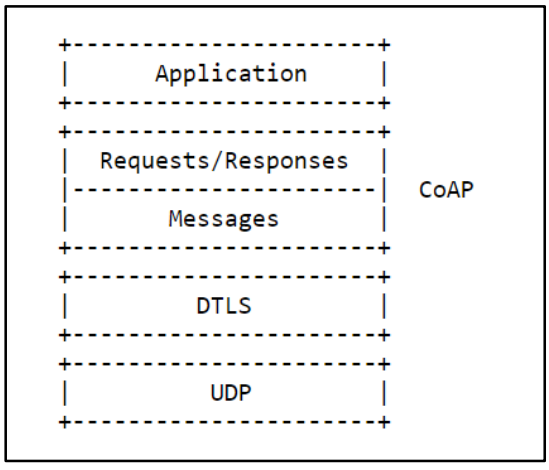


Figura 26.
Capa de seguridad. Abstraccion de CoAP
Fuente: RFC – 7252. https://datatracker.ietf.org/doc/html/rfc7252

El protocolo DTLS es una versión mejorada del ampliamente utilizado protocolo de seguridad de la capa de transporte (TLS) [RFC 5246]. La principal diferencia es que DTLS se ejecuta sobre UDP, en lugar de TCP, para proteger las principales aplicaciones UDP conocidas, como Voz sobre IP / Protocolo de inicio de sesión (VoIP / SIP). DTLS proporciona autenticación, integridad de datos, confidencialidad y gestión automática de claves (44)

CoAP define cuatro modos de seguridad para lograr la seguridad servicios requeridos (40). Estos modos son:

- **NoSec**: No hay seguridad a nivel de protocolo (DTLS está deshabilitado). Los paquetes se envían normalmente como datagramas UDP sobre IP y bajo el esquema indicado por CoAP (coap://). Las técnicas alternativas para proporcionar seguridad en la capa inferior deberían ser utilizadas cuando sea apropiado como (IPSec).
- **PreSharedKey:** DTLS está habilitado, hay una lista de claves previamente compartidas, y cada clave incluye una lista de los nodos que se pueden utilizarla para comunicarse. Si más de dos entidades comparten una clave específica, esta clave solo permite a las entidades autenticarse como miembros de ese grupo y no como un par específico.
- **RawPublicKey**: DTLS está habilitado y el dispositivo tiene un par de claves asimétricas sin ningún certificado. La clave se valida mediante un mecanismo OOB (Out-of-bound)
- **Certificate**: En este modo, DTLS está habilitado y el dispositivo tiene un par de claves asimétricas junto con un certificado X.509 que está firmado por una Root CA confiable y conocida.

CoAP en sí mismo no proporciona funcionalidades primitivas para autenticación o autorización; cuando sea necesario, puede ser proporcionado por seguridad de comunicación (es decir, IPsec o DTLS) o por seguridad de objeto (dentro de la carga útil) (40)

6 – Análisis comparativo. Conclusiones

Los cuatro protocolos de aplicación presentados son muy diferentes cada uno del otro. Han sido pensados para usos específicos disímiles y necesidades de mercado completamente distintas. Cada uno tiene fortalezas y debilidades, según sea el escenario donde se aplique y su uso depende del tipo de dispositivo IoT a implementar, los dispositivos a utilizar, los recursos con los cuales se cuente, entre otras cuestiones. Es propósito de este capítulo analizar las especificaciones de cada uno, a modo de resumen y comparar sus cualidades.

Arquitectura

Como se ha detallado tanto MQTT como AMQP son protocolos que tienen arquitectura de publicación / suscripción; en cambio CoAP y HTTP son protocolos cliente/servidor. Esto es relativo; dado que como se ha visto oportunamente CoAP tiene un modelo observador que permite la abstracción hacia un modelo de publicación / suscripción y HTTP en su versión /2 del protocolo tiene la funcionalidad Server que permite el envío proactivo de información. AMQP también permitirá actuar como cliente/servidor, al solicitar recursos a una cola sin permitir su consumo de manera automática.

Todos proveen semánticas bastantes simples y comunes a la hora de intercambiar información con un server o un bróker, según corresponda. MQTT pareciera ser el más simple y transparente, con cláusulas *Connect, Disconnect, Publish o Subscribe*. CoAP de manera implícita permite conexiones a la vez que solicita recursos, lo mismo que sucede con HTTP, utilizando GET. AMQP, por el contrario es quien presenta una estructura más compleja, propia de la naturaleza del protocolo y su arquitectura multicomponente, con métodos como Consume, Deliver o Get.

Mensajes

En cuanto a tamaño del mensaje y sobrecarga del mismo, CoAP es quien más pequeño tamaño de mensaje admite. Si bien es de tamaño indefinido, con un encabezado de 4 bytes, normalmente es lo suficientemente pequeño para ocupar como máximo un datagrama IP; dado que utiliza UDP como protocolo de transporte y tiene una baja sobrecarga sobre el mensaje. MQTT tiene un encabezado menor a CoAP (2 bytes) pero el protocolo utiliza TCP por cada uno de los mensajes, por lo que la sobrecarga aumenta y por lo tanto el tamaño del mensaje también. El encabezado de una trama AMQP es de 8 bytes, casi el doble que CoAP, pero también tiene *headers* extendidos y funcionalidades incorporadas que incrementan la sobrecarga; sumado a que utiliza TCP y sus correspondientes *handshakes*. Finalmente, HTTP resulta ser el protocolo más pesado de los cuatro, en cuanto a tamaño de mensaje: puede tener tantos headers como

se pretenda y el tamaño puede ser tan grande como se quiera negociar; dependiendo del servidor web donde se requiera el recurso. Considerando que HTTP fue diseñado originalmente para la navegación a través de sitios web e intercambio de grandes volúmenes de información; es lógico que requiera mayor cantidad de recursos que el resto de los protocolos.

Por supuesto, el tamaño del mensaje y el peso o sobrecarga del mismo también indica la cantidad de recursos que el dispositivo debería disponer para utilizar el protocolo. Mayor cantidad de recursos indicarían mayor consumo de energía. Tomando esta aseveración como verdadera, sin cambios dinámicos de red, para ambientes donde se requiera intercambios de mensajes pero a través de dispositivos restringidos, con pocos recursos y generalmente a batería (que requiere bajo consumo), los protocolos que utilizan mensajes de tamaño más pequeño y tienen menos sobrecarga, como CoAP y MQTT, parecerían ser los indicados. CoAP, incluso al usar UDP y asumir la no perdida de mensajes, debería tener menor consumo que el resto de los protocolos en estudio, según se especifica en estudios técnicos anteriores (45). AMQP requiere de procesos más complejos para la entrega de mensajes por lo que su consumo estaría más arriba que los protocolos anteriores. Finalmente, HTTP, para llevar adelante una operación de lectura de un recurso mediante GET, por ejemplo, necesita el envío de varios encabezados, más las conexiones TCP que lo transforman en el más demandante en cuanto a cantidad de procesamiento a requerir.

Características de Servicio

El propósito de este apartado es comparar las características propias de cada protocolo, buscando la beneficencia de uso de cada uno, su calidad de servicio, fiabilidad, interoperabilidad, entre otras cuestiones que resultan de gran relevancia a la hora de elegir un protocolo de comunicación para nuestra red IoT.

De los protocolos en estudio, quien más características de servicio ofrece es MQTT, que provee 3 calidades de servicio (QoS=0, QoS=1, QoS=2) y LWT ("última voluntad y testamento"); garantizando un estado a pesar de que el nodo se haya desconectado de manera inesperada. Además, al utilizar TCP pese a su carga, garantiza la entrega de paquetes con un protocolo de transporte fiable. AMQP, por su parte, define 2 niveles de servicio: entregado o no entregado. A diferencia de los dos protocolos anteriores CoAP no transporta mediante TCP por lo que pierde la fiabilidad de entrega del protocolo de transporte, pero busca compensar esto definiendo 2 tipos de mensajes para la entrega (similar a lo que define AMQP), pero que llama CON (mensaje confirmable) o NON (Mensaje no confirmable). Además, aporta dos características importantes que son mensajes de reconocimiento y reinicio; teniendo la posibilidad de retransmitir mensajes. Finalmente, HTTP no define ningún tipo de fiabilidad y deja todo en las manos del protocolo de transporte TCP, asumiendo su *delivery permanente.*

El despliegue interoperativo de los protocolos representa uno de los grandes desafíos de IoT, y es cuando HTTP "*toma la delantera"*. Es el más antiguo de los protocolos, el más probado

y ampliamente aceptado y para su funcionamiento solo necesita una pila HTTP; por lo que el montaje de un servidor o un cliente que utilice este protocolo es lo más sencillo. CoAP, al ser un protocolo con arquitectura request/response, permite una rápida adopción en dispositivos dado que es parte de la arquitectura web, utilizando incluso semántica similar; pero está limitada a dispositivos que utilicen UDP. AMQP, por su parte presenta la flexibilidad e interoperabilidad de poder utilizarse como protocolo request/response o pub/sub y con un payload negociable y sin límite; pero deben generarse necesariamente las colas de mensajería, los exchanges y lo referido al protocolo. Por último, MQTT solo permite utilización del protocolo de manera pub/sub, utilizando TCP y resultaría ser el que menos interoperabilidad logra de los cuatro en estudio.

Seguridad

En lo que respecta a la seguridad de los protocolos, HTTP y AMQP son los que más utilidades, funciones y operaciones brindan para resguardar los datos transmitidos. Por parte de HTTP se cuenta con dos métodos de autenticación que se detallaron oportunamente: Basic Authentication y HTTP Digest. Mientras que la primera utilizada user/password sin encriptar para autenticar un cliente, y por lo tanto debería utilizarse mediante una capa de encriptación en el transporte como TLS; la autenticación Digest usa encriptación para enviar user/password, sin la necesidad de TLS (aunque no es recomendado).

AMQP además de proporcionar TLS y diferentes formas de negociarlo, también proporciona SASL, agregando una capa más de seguridad y utilizando un framework conocido. Esto brinda confiabilidad en la utilización del protocolo.

MQTT ofrece el cifrado en el transporte, utilizando TLS, pero no ofrece ningún otro método de cifrado nativo o interno. Si bien brinda mecanismos para generar autorizaciones, mediante user/pass; si no se cifra a nivel de transporte, cualquier dato puede ser interceptado y leído. CoAP, por su parte, ofrece también cifrado a nivel de transporte con DTLS e IPSec, pero no brinda ninguna alternativa para autenticar internamente u otorgar autorizaciones mediante métodos primitivos.

Conclusiones

Este documento pretende analizar las características generales, la arquitectura, los mensajes y su sobrecarga, tamaño y estructura, y la seguridad de los protocolos IoT más utilizados, para poder identificar fortalezas y debilidades de cada uno.

MQTT es el protocolo más utilizado, según varias fuentes consultadas. Esto se da, básicamente por la fiabilidad del protocolo en la entrega, la simplicidad de su uso y las prestaciones como diferentes calidades de servicio (QoS) y LWT que ofrece. Resulta ser un protocolo liviano, con transporte fiable y características interesantes. MQTT descansa la seguridad en TLS, estableciendo la responsabilidad en el transporte en un protocolo ampliamente probado y documentado.

AMQP es más complejo y es el menos utilizado de los protocolos abarcados en este estudio, según el cuadro comparativo establecido. El protocolo nace con fines comerciales y busco brindar fiabilidad e interoperabilidad; características que, según la evaluación en este trabajo, ha alcanzado satisfactoriamente. Es, posiblemente, el más seguro de los cuatro protocolos en estudio y su utilización deba darse acorde a escenarios donde el manejo de información, su confidencialidad y la autorización sean transcendentales.

CoAP es el protocolo más liviano de los estudiados. Es muy versátil, ligero y de bajo consumo. Ha encontrado la forma de solucionar (en parte) la poca fiabilidad que brinda UDP mediante funcionalidades como reconocimiento y reinicio, además de mensajes confirmables y no-confirmables. Trabaja de forma segura y brinda la posibilidad de trabajar en modo observador para, entre otras cuestiones, generar ahorro de energía, conectividad y demás consumos en cliente. Resulta un protocolo muy interesante para utilizar en dispositivos de bajos recursos y en ámbitos donde no sea necesario garantizar la totalidad de la recepción y envío de los mensajes.

HTTP es largamente el más antiguo y bastamente probado de los protocolos en estudio. Tiene la ventaja del conocimiento amplio de la comunidad en la implementación y la facilidad de manejo. El despliegue inter-operativo es uno de sus fuertes. Tiene parte del mercado ganado por su rápida implementación y gran conocimiento. Solo es necesario una pila HTTP para implementarlo. Sus contras radican en la gran cantidad de recursos que tienen que tener los dispositivos para su utilización óptima; algo que es muy poco frecuente en los dispositivos autónomos IoT. La seguridad sobre HTTP, utilizando TLS y Basic Authentication o HTTP Digest, es ampliamente conocida y documentada. Con la llegada de HTTP/2 el protocolo busca comprimir encabezados, optimizando la performance de transferencia. Sin lugar a dudas HTTP es un protocolo a tener en cuenta para el despliegue de redes de dispositivos IoT que no tengan restricciones de procesamiento, almacenamiento o transferencia.

La forma en que se ha desarrollado la arquitectura de IoT en los últimos años se manifiesta en la cantidad de opciones de plataformas y protocolos disponibles. El desafío, desde el aspecto del diseño, para lograr interoperabilidad, correcto funcionamiento de los dispositivos y seguridad en la información que estos almacenan, procesan o transmiten, reside en garantizar

que se utilice el protocolo correcto en el ambiente correcto, siguiendo las prácticas recomendadas y obviando las configuraciones por defecto. Se deben considerar todas las herramientas que los protocolos ponen a disposición para realizar una comunicación más eficiente y segura, tomando como referencia el costo/beneficio de cada implementación, según la naturaleza y necesidad de uso de la solución tecnológica.

Más allá de la mejora en algunos aspectos técnicos de los estándares, que evolucionan de manera continua, el reto se encuentra en la elección del protocolo y el correcto diseño de la solución, según los recursos disponibles, el tipo de información a transmitir, su sensibilidad y operatividad, la cantidad de clientes que utilizarían el / los recursos y el entorno en el que se encuentren los equipos.

Bibliografía

1. **Internet Society.** Internet Society . [En línea] [Citado el: 15 de Marzo de 2019.] https://www.internetsociety.org/iot.

2. **IHS .** IoT platforms: enabling the Internet of Things. [En línea] [Citado el: 19 de Febrero de 2019.] https://cdn.ihs.com/www/pdf/enabling-IOT.pdf.

3. **How IoT impacts data and analytics. [En línea] Gartner. [Citado el: 15 de Febrero de 2019.] https://www.gartner.com/smarterwithgartner/how-iot-impacts-data-and-analytics/.**

4. Wall Street Daily - The Best Play on the Internet of Things Trend. [En línea] [Citado el: 15 de Marzo de 2019.] https://www.wallstreetdaily.com/2015/12/21/internet-of-things-future/.

5. ArcServe IoT Security Issues. [En línea] [Citado el: 12 de Marzo de 2019.] https://www.arcserve.com/insights/iot-security-issues/.

6. Leading the IoT - How to lead in a connected world. [En línea] [Citado el: 28 de Marzo de 2019.] https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf.

7. Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. [En línea] [Citado el: 2019 de Marzo de 19.] https://www.rfc-editor.org/rfc/rfc7452.txt.

8. Noura, Mahda, Atiquzzaman, Mohammed y Gaedke, Martin. Interoperability in Internet of Things: Taxonomies and Open Challenges. [En línea] [Citado el: 03 de Junio de 2019.] https://link.springer.com/article/10.1007/s11036-018-1089-9.

9. Oracle. Oracle - Protocolos de capa de aplicacion. [En línea] [Citado el: 30 de Mayo de 2019.] https://docs.oracle.com/cd/E19957-01/820-2981/ipov-22/index.html.

10. Publisher-Subscriber pattern. [En línea] [Citado el: 2019 de abril de 15.] https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber.

11. Amazon Web Services - pub/sub. [En línea] 15 de Febrero de 2019. https://aws.amazon.com/es/pub-sub-messaging/.

12. MQ Telemetry Transport - Mosquito Eclipse Foundation. [En línea] [Citado el: 23 de mayo de 2019.] https://mosquitto.org/man/mqtt-7.html.

13. Universidad de Valladolid. El modelo cliente/servidor - Sistemas Distribuidos. [En línea] https://www.infor.uva.es/~fdiaz/sd/2005_06/doc/SD_TE02_20060305.pdf.

14. MQTT Protocol. [En línea] 26 de Marzo de 2019. http://mqtt.org/.

15. MQTT OASIS Standard . [En línea] 6 de mayo de 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

16. IBM. QoS MQTT. [En línea] [Citado el: 07 de Julio de 2019.] https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q029090_.htm.

17. HiveMQ - Fundamentos de MQTT. [En línea] [Citado el: 26 de Marzo de 2019.] https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

18. ISO/IEC 20922:2016 - Message Queuing Telemetry Transport (MQTT) v3.1.1. [En línea] 20 de mayo de 2019. https://www.iso.org/standard/69466.html.

19. Seguridad de MQTT. [En línea] 29 de 05 de 2019. https://www.ibm.com/support/knowledgecenter/es/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc00150_.htm.

20. *AMQP Standar .* [En línea] [Citado el: 26 de mayo de 2019.] http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.

21. AMQP.org . [En línea] [Citado el: 30 de Marzo de 2019.] http://www.amqp.org/about/what.

22. Department of Computer Science and Engineering IIT Bombay. [En línea] [Citado el: 23 de Marzo de 2020.] https://www.cse.iitb.ac.in/~comad/2010/pdf/Industry%20Sessions/UID_Pramod_Varma.pdf.

23. AMQP - examples. [En línea] [Citado el: 23 de Marzo de 2020.] https://www.amqp.org/about/examples.

24. Ted Ross. Integrating the Internet of Things with AMQP. [En línea] [Citado el: 23 de Marzo de 2020.] https://es.scribd.com/document/234021715/AMQP-IoT-pdf .

25. OASIS. *A Comparison of AMQP and MQTT.* [En línea] [Citado el: 23 de abril de 2021.] https://lists.oasis-open.org/archives/amqp/201202/msg00086/StormMQ_WhitePaper_-_A_Comparison_of_AMQP_and_MQTT.pdf.

26. Boris Adryan, Dominik Obermaier, Paul Fremantle. *The Technical Foundations of IoT.* 2017. 9781630812515.

27. *AMQP documentation - Rabbit AMQP.* [En línea] [Citado el: 03 de Marzo de 2020.] https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf.

28. AMQP: conoce el Advanced Message Queuing Protocol. [En línea] https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/advanced-message-queuing-protocol-amqp/.

29. OASIS. AMQP Stantadar. [En línea] [Citado el: 18 de Marzo de 2020.] http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.

30. AMQP - Diseño de arquitectura: comunicación entre sistemas. [En línea] https://blog.csdn.net/yinwenjie/article/details/50820369.

31. OASIS Standard. OASIS Advanced Message Queuing Protocol (AMQP) - Security. [En línea] [Citado el: 29 de Mayo de 2021.] https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-security-v1.0.html.

32. APACHE QPID. AMQP Messaging Broker - Security. [En línea] https://qpid.apache.org/releases/qpid-cpp-1.39.0/cpp-broker/book/chap-Messaging_User_Guide-Security.html.

33. IBM . IBM MQ - Proteccion de clientes AMQP. [En línea] https://www.ibm.com/docs/es/ibm-mq/9.0?topic=ssfksj-9-0-0-com-ibm-mq-sec-doc-amqp-security-htm.

34. Mozilla org. Generalidades del protocolo HTTP. [En línea] [Citado el: 27 de Febrero de 2020.] https://developer.mozilla.org/es/docs/Web/HTTP/Overview.

35. IBM. Servicios Web de RESTful: Los aspectos básicos. [En línea] [Citado el: 11 de Febrero de 2020.] https://developer.ibm.com/es/articles/ws-restful/.

36. AWS - Amazon Web Services. AWS IoT Core - Developer Guide. [En línea] https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf.

37. Mozilla. MDN Web Docs - Mensajes HTTP. [En línea] [Citado el: 23 de Marzo de 2021.] https://developer.mozilla.org/es/docs/Web/HTTP/Messages.

38. O´Reilley. High Performance Browser Networking - Transport Layer Security (TLS). [En línea] [Citado el: 23 de mayo de 2021.] https://hpbn.co/transport-layer-security-tls/.

39. CoAP. CoAP Technology. [En línea] [Citado el: 16 de Enero de 2020.] https://coap.technology/ .

40. Internet Engineering Task Force (IETF). The Constrained Application Protocol (CoAP). *CoAP - RFC7252.* [En línea] 2014. [Citado el: 16 de Febrero de 2020.] https://datatracker.ietf.org/doc/html/rfc7252. ISSN: 2070-1721.

41. Parra, Carlos A. Hervas. SEDICI - UNLP. *Análisis de rendimiento de protocolos .* [En línea] 2018. http://sedici.unlp.edu.ar/bitstream/handle/10915/69435/Documento_completo.pdf-PDFA.pdf.

42. Internet Engineering Task Force (IETF) . *Observing Resources in the Constrained Application Protocol (CoAP) - RFC 7641.* [En línea] 2015. [Citado el: 25 de febrero de 2020.] https://datatracker.ietf.org/doc/html/rfc7641. ISSN: 2070-1721.

43. Heredia, Jorge Castro. Uso del protocolo CoAP para la implementación de una aplicación domótica con redes de sensores inalámbricas. [En línea] https://repositorio.upct.es/bitstream/handle/10317/4163/pfc5908.pdf.

44. A. Lasebae, Mahdi Aiash. *Security Analysis of the Constrained Application Protocol.* [En línea] [Citado el: 24 de Febrero de 2020.] https://www.researchgate.net/publication/259869307_Security_Analysis_of_the_Constrained_Application_Protocol_in_the_Internet_of_Things.

45. *Performance evaluation of MQTT and CoAP via a common middleware.* D. Thangavel, X. Ma, A. Valera, H. Tan and C. K. Tan. s.l. : EEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014. doi: 10.1109/ISSNIP.2014.6827678.

46. Rose, Karen, Eldridge, Scott y Chapin, Lyman. Internet Society. [En línea] 6 de Diciembre de 2018. https://www.internetsociety.org/es/resources/doc/2015/iot-overview.

47. Internet de las cosas, una breve reseña. [En línea] [Citado el: 05 de Marzo de 2019.] https://www.internetsociety.org/wp-content/uploads/2017/09/report-InternetOfThings-20160817-es-1.pdf.

48. International Business Machines Corporation (IBM). Protocolo MQTT V3.1. [En línea] [Citado el: 07 de julio de 2019.] http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

49. *A comparative evaluation of AMQP and MQTT protocolsover unstable and mobile networks.* J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, P . Manzoni. 2th Annual IEEE ConsumerCommunications and Networking Conference : s.n., 2015.

50. Chen, Xi. Constrained Application Protocol for Internet of Things. [En línea] [Citado el: 21 de Noviembre de 2020.] https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/.

51. Internet Engineering Task Force (IETF) . Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. [En línea] [Citado el: 23 de 01 de 2020.] https://datatracker.ietf.org/doc/html/rfc7231#section-4.

52. Lodoño, Diego y Cespedes, Sandra. Performance Evaluation of CoAP and HTTP/2 in Web Applications. [En línea] http://ceur-ws.org/Vol-1727/ssn16-final5.pdf.

53. Auto-id Labs. Advancing the IoT for Global Commerce. [En línea] [Citado el: 12 de Febrero de 2019.] https://www.autoidlabs.org/.

54. Yuan, Michael. IBM: Conociendo MQTT. [En línea] 26 de Marzo de 2019. https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html.