Revisión literaria de protocolos de capa de aplicación para Internet de las Cosas en redes IEEE 802.15.4

Diana Milena Flórez Mantilla

Iris Rocío Ortiz Cáceres

Teodomiro José Fajardo Zárate

Trabajo de Grado para Optar el título de Especialista en Telecomunicaciones

Director

Pedro Javier Trujillo Tarazona

Magister en Informática

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Ingeniería Eléctrica Electrónica y Telecomunicaciones

Especialista en Telecomunicaciones

Bucaramanga

2020

Agradecimientos

A Dios por guiarnos a lo largo de nuestra vida, brindándonos paciencia y sabiduría para culminar con éxito nuestras metas propuestas.

A nuestras familias por su apoyo incondicional y estar siempre motivándonos.

A la Universidad Industrial de Santander, por permitirnos ser parte de esta prestigiosa institución.

A nuestro director Mg. Pedro Javier Trujillo Tarazona, quien con su conocimiento, acompañamiento y apoyo incondicional nos orientó en el desarrollo de este trabajo.

A cada uno de los docentes, por toda su orientación y todo lo aprendido durante nuestra formación para poder optar por el título de Especialistas en Telecomunicaciones.

A nuestros compañeros que con su dedicación, apoyo, entrega y paciencia hicieron posible alcanzar este objetivo.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Glosario

Apogee. Apogee Instruments, es un fabricante de sensores ambientales.

Contiki. Es un sistema operativo de código abierto para el Internet de las cosas.

Decagon. Decagon Devices, es una compañía que diseña, produce y comercializa instrumentos científicos agrícolas que se centra en la medición de agua, luminosidad y calor en el suelo, plantas y atmósfera ("Decagon Devices América Latina Ltda.", 2020).

Ecomatik. Es una compañía que se dedica a desarrollar y producir instrumentos electrónicos para mediciones continuas del crecimiento de plantas y frutos, el consumo de agua de las plantas y el agua del suelo disponible para las plantas ("Home - Ecomatik Umweltmess- und Datentechnik", 2016).

Gill Instruments. Empresa dedicada al suministro de anemómetros ultrasónicos para medir la velocidad y dirección del viento, junto con accesorios para el viento ("Gill Instruments - About us", 2020).

IoE. Internet of Everything. Es la conexión de personas, datos, procesos y cosas con el Internet a través del procesamiento de información.

Libellium. Es una multinacional tecnológica española, que diseña y fabrica hardware y software para redes de sensores inalámbricos. (Libelium, 2018a)

LLN Low-Power and Lossy Networks. Las redes de baja potencia y con pérdidas, son aquellas que están conformadas de numerosos dispositivos integrados y sensores, cuentan con recursos limitados para su funcionamiento. (Vasseur, 2014).

RFC Request for Comments. Son publicaciones del grupo de trabajo de Ingeniería de Internet (IETF) que contienen notas técnicas sobre el funcionamiento de Internet y otras redes de computadores, incluyendo protocolos, procedimientos, programas y nuevas tecnologías. (IETF, 2018).

Smart Grid. La red eléctrica inteligente es una tecnología de información y comunicación que permite un sistema moderno de generación, transmisión, distribución y consumo de electricidad (Sethi & Sarangi, 2017).

TelosB. Es una plataforma de código abierto diseñada para permitir la experimentación en redes de sensores inalámbricos ("TelosB", s/f).

TinyOS. Es un sistema operativo de código abierto, diseñado para dispositivos inalámbricos de baja potencia, como los que se utilizan en redes de sensores, computación ubicua, redes de área personal, edificios y medidores inteligentes ("TinyOS", s/f).

3GPP. 3rd Generation Partnership Project. Es una tecnología desarrollada para permitir que dispositivos de Internet de las cosas puedan conectarse directamente a una red 4G (González García, 2017).

QUIC Quick UDP Internet Connections. Es un protocolo de transporte sobre UDP desarrollado por Google, utiliza mecanismos de TCP e introduce nuevas características no utilizadas por otros protocolos de transporte. (Gratzer, 2016)

Abreviaturas

AES: Advanced Encryption Standard

ALPN: Application Layer Protocol Negotiation

AMCA: Asynchronous Multi-Channel Adaption

AMQP: Advanced Message Queuing Protocol

API: Application Programming Interface

BSD: Berkeley Software Distribution

CoAP: Constrained Application Protocol

CSL: Coordinated Sampled Listening

CSMA: Carrier Sense Multiple Access

CSMA-CA: Carrier Sense Multiple Access with Collision Avoidance

DAG: Directed Acyclic Graph

DES: Data Encryption Standard

DODAG: Destination-Oriented DAG

DSME: Deterministic & Synchronous Multi-channel Extension

DSSS: Direct Sequence Spread Spectrum

DTLS: Datagram Transport Layer Security

ED: Energy Detection

EDL: Eclipse Distribution License

EPL: Eclipse Public License

FFD: Full Function Devices

GPL: Licencia Pública General de GNU

GTS: Guaranteed Time Slot

HTTP/2: (Hypertext Transfer Protocol Version 2

IETF: Internet Engineering Task Force

IEEE: Institute of Electrical and Electronics Engineers

IoT: Internet of Things

IPsec: Internet Protocol Security

IPv6: Internet Protocol Version 6

LE: Low Energy

LGPL: GNU Lesser General Public License

LLDN: Low Latency Deterministic Network

LQI: Link Quality Indicator

LR-WPAN: Low-Rate Wireless Personal Area Network

LWIP: lightweight IP

M2M: Machine to Machine

mDNS: Multicast Domain Name SystemMini

MIT: Massachusetts Institute of Technology

MLME: MAC Sublayer Management Entity

MQTT: Message Queue Telemetry Transport

MQTT-SN: Message Queue Telemetry Transport-Sensor Network

MTU: Maximum Transmission Unit

NFC: Near Field Communication

NPN: Next Protocol Negotiation

PAN: Personal Area Network

QUIC: Quick UDP Internet Connections

RFD: Reduced Function Devices

RFID: Radio Frequency Identification

RIT: Receiver Initiated Transmissions

ROLL: Routing Over Low Power and Lossy Networks

RPL: Routing Protocol for Low-Power and Lossy Networks

RTT: Round-Trip Time o Round-Trip Delay Time

SAP: Service Access Point

SASL: Simple Authentication and Security Layer

SSL: Secure Sockets Layer

TCP: Transmission Control Protocol

TLS: Transport Layer Security

TSCH: Time Slotted Channel Hopping

UDP: User Datagram Protocol

URI: Uniform Resource Identifier

Wi-SUN: Wireless Smart Metering Utility Network

WSN: Wireless sensor networks

XMPP: Extensible Messaging and Presence Protocol

XML: Extensible Markup Language

6LoWPAN: IPv6 over Low Power Wireless Personal Area Networks

Resumen

Título: Revisión literaria de protocolos de capa de aplicación para Internet de las Cosas en redes IEEE 802.15.4[*]

Autores: Diana Milena Flórez Mantilla[**]
Iris Rocío Ortiz Cáceres
Teodomiro José Fajardo Zárate

Palabras Clave: LLN, IEEE 802.15.4, IoT**,** CoAP, MQTT, AMQP, WebSocket, XMPP, HTTP/2.

Descripción:

En este trabajo se presenta una revisión literaria de protocolos de capa de aplicación de Internet de las cosas en redes LLN de tipo IEEE 802.15.4, se inicia con la descripción de las características de estas redes de baja potencia y con pérdidas, asociadas a IoT. Posteriormente se toma como referencia una arquitectura de 5 niveles, que describe de forma conceptual la lógica de la tecnología IoT sobre IEEE 802.15.4, usando TCP/IP. En la capa superior de esta arquitectura, encontramos los protocolos de capa de aplicación, como son: CoAP, MQTT, AMQP, WebSocket, XMPP y HTTP/2, donde se describen los aspectos de la estructura y funcionamiento para cada uno, permitiendo comprender su modelo de comunicación para Internet de las cosas; además de atributos como la estructura del mensaje, la seguridad, las características extendidas que ofrecen otras funcionalidades al protocolo. Así mismo, se presenta una descripción de algunas implementaciones desarrolladas para cumplir las funcionalidades de los protocolos. También, se presentan casos de aplicación de protocolos de capa de aplicación para IoT en áreas como: hogares inteligentes, edificios inteligentes, ciudades inteligentes, industria, energía, ambiente inteligente, agricultura y salud. Por último, se describen aportes de diversos autores, relacionados con la seguridad para cada uno de los protocolos.

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Eléctrica Electrónica y Telecomunicaciones. Especialista en Telecomunicaciones. Director: Pedro Javier Trujillo Tarazona, Magister en Informática

Abstract

Title: Revisión literaria de protocolos de capa de aplicación para Internet de las Cosas en redes IEEE 802.15.4*

Authors: Diana Milena Flórez Mantilla**
Iris Rocío Ortiz Cáceres
Teodomiro José Fajardo Zárate

Key Words: LLN, IEEE 802.15.4, IoT**,** CoAP, MQTT, AMQP, WebSocket, XMPP, HTTP/2.

Description:
This paper presents a literary review of Internet of Things application layer protocols in IEEE 802.15.4 type LLN networks, starting with the description of the characteristics of these Low-Power and Lossy Networks associated with IoT. Subsequently, a 5-level architecture is taken as a reference, which conceptually describes the logic of IoT technology over IEEE 802.15.4, using TCP/IP. In the upper layer of this architecture we find the application layer protocols, such as: CoAP, MQTT, AMQP, WebSocket, XMPP and HTTP/2, where aspects of the structure and operation for each are described, allowing us to understand their model communication for the Internet of things; in addition to attributes such as message structure, security, and extended features that the protocol offers other functionalities. Likewise, a description of some implementations developed to fulfill the functionalities of the protocols is presented. In addition, application cases of application layer protocols for IoT are presented in areas such as: smart homes, smart buildings, smart cities, industry, energy, smart environment, agriculture and health. Finally, contributions from other authors, related to security for each of the protocols, are described.

* Degree work

** Faculty of Physicomechanical Engineering. School of Electrical, Electronic and Telecommunicactions Engineering. Specialization in Telecommunications. Director: Pedro Javier Trujillo Tarazona, Master in Informatics

Introducción

IoT es un concepto de utilización de Internet, consistente en la interconexión de objetos y entornos personales a través de distintas redes de comunicación para compartir información y tomar decisiones de forma independiente (Rose, Karen; Eldridge, Scott; Chapin, 2015).

En algunos contextos, IoT requiere tecnologías como las redes LLN (Low-Power and Lossy Network) que se ajustan a las necesidades de ciertos casos de uso, donde se requiere establecer comunicaciones de bajo costo utilizando dispositivos de bajo consumo. En este documento, el enfoque principal se dirige hacia las funciones que presta la capa de aplicación de IoT, para lo cual, profundizaremos en los diferentes protocolos de esta capa para redes LLN, que satisfacen las necesidades de los usuarios accediendo a los servicios que ellos prestan.

Con esta revisión literaria de protocolos de capa de aplicación para IoT en redes IEEE 802.15.4, se busca adquirir información pertinente y actualizada, que describa los aspectos más importantes de los protocolos de capa de aplicación, tales como: arquitectura, funcionamiento, características, limitaciones y casos de aplicación.

Este documento está organizado en tres partes. En la primera parte se muestra el planteamiento y la justificación del problema en el capítulo uno, donde se expone el tema a abordar y las razones por las cuales se realiza la revisión literaria de protocolos de capa de aplicación para Internet de las Cosas en redes IEEE 802.15.4; se presentan luego los objetivos en el capítulo dos; y en el capítulo tres, se exponen algunos antecedentes de estudios realizados a los protocolos de la capa de aplicación para Internet de las cosas en redes IEEE 802.15.4.

En la segunda parte, se abarcan los capítulos del cuatro al seis, donde se desarrolla la temática central de este reporte. En el capítulo cuatro se detallan las principales características de las

redes de baja potencia y con pérdidas (LLN, Low-Power and Lossy Networks) de tipo IEEE 802.15.4. Posteriormente, en el capítulo cinco, se describe una arquitectura de protocolos del Internet de las cosas en redes LLN. Finalmente, en el capítulo seis, se describen las características, arquitectura y seguridad de los protocolos de capa de aplicación para IoT en las redes de baja potencia y con pérdidas de tipo IEEE 802.15.4.

La última parte corresponde a las conclusiones, las recomendaciones y las referencias bibliográficas.

1. Planteamiento y justificación del problema

IoT o Internet de las Cosas nace de la necesidad de estar conectados entre todos, para enviar o recibir información, de tal manera que un individuo o dispositivo inteligente pueda comunicarse de forma remota con cualquier objeto, compartir información y tomar decisiones de forma autónoma. Algunos de los escenarios donde se aplica Internet de las cosas son: la salud, la domótica, la educación, la industria, el transporte, la agricultura, el medio ambiente, los servicios de energía y la seguridad entre otros.

En estas áreas, una de las tecnologías que ha tenido mayor éxito son las redes de sensores inalámbricos debido a la capacidad de recolectar diferentes tipos de datos para luego ser monitorizados y gestionados, lo que ha permitido aumentar su popularidad en la investigación industrial y académica (Rosero et al., 2017). Este tipo de redes incluyen las que aplican el estándar 802.15.4 del IEEE (Institute of Electrical and Electronics Engineers) y generalmente son denominadas redes LLN (Low-Power and Lossy Networks), las cuales presentan limitaciones en cuanto a la energía, la memoria, los recursos de procesamiento y el ancho de

banda. Además, los dispositivos de estas redes inalámbricas tienen una naturaleza heterogénea dado que están conformados por tecnologías, protocolos y estándares diferentes, generando incompatibilidad entre ellos.

Sin embargo, aunque los dispositivos embebidos inteligentes se transformen en parte importante de IoT, su integración aun presenta varios desafíos, debido a que muchas tecnologías de Internet y protocolos existentes no fueron diseñadas para esta clase de dispositivos. Inicialmente las arquitecturas y protocolos propietarios se enfocaron en extender tecnologías de Internet como IoT a estos dispositivos limitados, luego el IETF (Internet Engineering Task Force), una organización internacional de normalización abierta se unió a la integración de dispositivos con restricciones en Internet produciendo protocolos estandarizados abiertos basados en IP (Internet Protocol) (Ishaq et al., 2013).

IoT tiene como objetivos principales asegurar una eficaz comunicación entre objetos y construir un vínculo continuo entre ellos, utilizando diferentes tipos de aplicaciones (Yassein, Shatnawi, & Al-Zoubi, 2016). Para lograr estas metas IoT usa su capa de aplicación, donde su importancia radica en proporcionar servicios inteligentes de alta calidad, para satisfacer las necesidades de los clientes (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015); reduciendo el esfuerzo humano y mejorando la calidad de vida (Sethi & Sarangi, 2017). Los protocolos de capa de aplicación en el entorno IoT, permiten a los usuarios interactuar con los objetos conectados y determinan la manera como se organizan los datos que van a ser enviados a los nodos del sistema (Yassein et al., 2016) (Al-Fuqaha et al., 2015).

Además, los protocolos de la capa de aplicación toman gran importancia, ya que permiten el acceso de los usuarios a la información que manejan los dispositivos conectados (Yassein et al., 2016) (Al-Fuqaha et al., 2015), pueden reducir el tráfico de red y mejorar la confiabilidad, como

se puede evidenciar en investigaciones como en (Sharma & Gondhi, 2018), (Tandale, Momin, & Seetharam, 2017). La elección del protocolo depende de las necesidades de cada caso de aplicación.

Diferentes organizaciones vienen trabajando en el desarrollo de protocolos que permitan la interoperabilidad de dispositivos en múltiples escenarios (Saritha & Sarasvathi, 2017). Existen protocolos de la capa aplicación en IoT que implementan diferentes características en redes restringidas como en (Sharma & Gondhi, 2018) donde se analiza la eficacia y la confiabilidad sobre la base de la eficiencia energética, la seguridad y la naturaleza liviana de los protocolos y en (Novelli, Jorge, Melo, & Koscianski, 2018), se presentan características como arquitectura, calidad de servicio (QoS), mecanismos de seguridad, recursos de servicio de descubrimiento y opciones de integración web, entre otros, en diferentes tipos de redes inalámbricas.

Por lo que toma importancia revisar los protocolos de capa de aplicación para IoT en redes de baja potencia y con pérdidas de tipo IEEE 802.15.4, para adquirir información pertinente y actualizada, que permita a los autores adentrarse en el tema para su posterior aplicación, ampliar la visión de desarrolladores de software en IoT para redes inalámbricas y sirva de fundamento para futuras investigaciones.

2. Objetivos

2.1 Objetivo general

Realizar una revisión literaria de protocolos de capa de aplicación para Internet de las cosas (IoT) en redes IEEE 802.15.4.

2.2 Objetivos específicos

1. Describir las características fundamentales de las redes de baja potencia y con pérdidas (LLN, Low-Power and Lossy Network) de tipo IEEE 802.15.4.
2. Describir la arquitectura de protocolos para Internet de las Cosas en redes de baja potencia y con pérdidas (LLN, Low-Power and Lossy Network) de tipo IEEE 802.15.4.
3. Describir elementos relevantes de la estructura y funcionamiento de protocolos de aplicación para Internet de las Cosas (IoT: Internet of Things) o Internet de Todo (IoE: Internet of Everything) tales como CoAP (Constrained Application Protocol, RFC 7252, 2014), MQTT (Message Queue Telemetry Transport, ISO/IEC 20922:2016), AMQP (Advanced Message Queuing Protocol, ISO/IEC 19464:2014), WebSocket (RFC 6455, 2011), XMPP (Extensible Messaging and Presence Protocol, RFC 6120, 2011) y HTTP/2 (Hypertext Transfer Protocol Version 2, RFC 7540, 2015).
4. Presentar implementaciones, con sus especificaciones o características, de protocolos de capa de aplicación para IoT o IoE (Internet of Everything).

5. Presentar casos de aplicación de protocolos de capa de aplicación para IoT e IoE en redes de baja potencia y con pérdidas “LLN” en redes inalámbricas IEEE 802.15.4.

3. Antecedentes

El Internet de las cosas nació de la necesidad de monitorizar y controlar ambientes por medio del Internet, desde cualquier lugar con el uso de tecnologías inteligentes a través de la interconexión de objetos capaces de recolectar, transmitir información e interpretarla para un beneficio común.

Ha pasado más de una década desde que fue utilizado el termino IoT. Sin embargo, a pesar de los esfuerzos de grupos de investigación e innovadoras corporaciones, todavía hoy no es posible decir que IoT está ampliamente adoptado. Esto se debe principalmente al hecho de que una arquitectura de IoT unificada aún no se ha definido claramente y no existe un acuerdo en común en los protocolos y estándares para todas las capas de IoT (Ramirez & Pedraza, 2017).

A nivel nacional, en (Ramirez & Pedraza, 2017) se presentó un análisis de rendimiento en los protocolos de comunicación para Internet de las cosas en el departamento de sistemas de la Universidad Nacional de Colombia, para determinar la carga computacional, gastos generales y ancho de banda de red de los protocolos MQTT, AMQP, CoAP y XMPP.

A nivel internacional se han presentado diversas investigaciones con el propósito de profundizar y presentar documentos que sirvan de soporte a profesionales o entidades que requieran hacer implementaciones en IoT; los cuales se presentan a continuación:

En (Naik, 2017), se presentó una evaluación de los cuatro protocolos de mensajería emergentes en sistemas de IoT: MQTT, CoAP, AMQP y HTTP, para obtener una idea de sus

fortalezas y limitaciones, permitiendo que el usuario pueda decidir su uso relevante en sistemas IoT, basados en sus requisitos e idoneidad.

En algunos sectores beneficiados por IoT como en: la agricultura, la industria, la salud, la automatización, entre otras, se requiere gran capacidad de almacenamiento y computo; en el documento (Chaudhary, Peddoju, & Kadarla, 2017) se describió un estudio realizado a los protocolos CoAP, MQTT y AMQP de Internet de las cosas IoT, usados para intercambiar datos con fuentes externas.

Otras investigaciones se centraron en hacer comparaciones entre protocolos de capa de aplicación, en (Mijovic, Shehu, & Buratti, 2016) se evalúa el rendimiento de tres protocolos: CoAP, WebSocket y MQTT, para demostrar cual protocolo es el de mayor eficiencia y con el promedio más bajo de tiempo de ida y vuelta RTT (round-trip time o round-trip delay time).En (Karagiannis, Chatzimisios, Vazquez-Gallego, & Alonso-Zarate, 2015) se demuestra un análisis comparativo de los protocolos MQTT, CoAP, AMQP y HTTP en cuanto a la calidad de los servicios e interoperabilidad. En (Kayal & Perros, 2017) se presentó un sistema de parqueo inteligente utilizando los protocolos COAP, MQTT, XMPP y WebSocket con el fin de medir y comparar su tiempo de respuesta para diferentes cargas.

En varias conferencias se profundizó en la capa de aplicación de Internet de las cosas y sus protocolos. En (Yassein et al., 2016) se realiza una breve descripción de las investigaciones más recientes sobre los protocolos XMPP, MQTT, CoAP, DSS, AMQP y WebSocket, la cual se centró en la valuación de cada protocolo en términos de la arquitectura, modelo de comunicación, la seguridad, y el logro de la calidad de los servicios.

En (Karagiannis et al., 2015) se presentó una comparativa entre los protocolos de aplicación, en los que se destacó IETF CoAP, de IBM (International Business Machines) MQTT,

WebSocket 5s HTML entre otros; se argumentó acerca de su idoneidad en IoT considerando aspectos de fiabilidad, seguridad y consumo de energía.

En (Al-Fuqaha et al., 2015) se presenta descripción general de algunos detalles técnicos sobre protocolos CoAP, MQTT, AMQP, XMPP, DDS (Data Distribution Service), mDNS (Multicast Domain Name System), comparados con otros trabajos de conferencias, aportando un resumen completo de los protocolos y aplicaciones más relevantes permitiendo a desarrolladores de aplicaciones actualizarse rápidamente.

Por otro lado, cabe mencionar una de las tecnologías de mayor éxito en el campo del IoT, como son las redes de sensores inalámbricos, debido a que permiten recoger diferentes tipos de datos, procesarlos para luego poder ser monitorizados y gestionados (Rosero et al., 2017). Se demuestran en algunas aplicaciones como: En el proyecto (Castro Heredia, 2014) se describe el desarrollo de una aplicación utilizando sensores de bajo consumo (TelosB) con el protocolo de aplicación CoAP para un Smart Home. En (Kayal & Perros, 2017), se describió la implementación de una aplicación IoT para estacionamientos inteligentes, con el uso y comparativa de los protocolos COAP, MQTT, XMPP y WebSocket.

En la implementación (Kuladinithi, Bergmann, Thomas Pötsch, Becker, & Görg, 2011), se presenta una solución para integrar las redes de sensores utilizados en los procesos logísticos de un contenedor de carga marítima, evaluado sobre los sistemas operativos TinyOS y Contiki. En el mercado actual grandes empresas se han destacado por la implementación de sistemas inteligentes, en el artículo (Libelium, 2018a), se describe la nueva plataforma IoT para la agricultura inteligente de Libelium, una solución que incluye 19 tipos de sensores de los más prestigiosos y fiables fabricantes de tecnología agrícola como Apogee Instruments, Decagon, Ecomatik y Gill Instruments.

4. Las redes de baja potencia y con pérdidas de tipo IEEE 802.15.4

Las redes LLN o redes de baja potencia y con pérdidas, son aquellas que están compuestas de numerosos dispositivos integrados y cuentan con recursos limitados para su funcionamiento como: bajo consumo de energía, bajo procesamiento de cómputo y poca cantidad de memoria. Estos dispositivos se conectan por una variedad de enlaces, basados en el estándar IEEE 802.15.4 o WiFi de baja potencia (Vasseur, 2014).

Las redes LLN se pueden aplicar en diferentes campos como son: Smart Grid (Infraestructura de medición avanzada en las redes inteligentes), redes de sensores urbanas (monitoreo del medio ambiente), sistemas de monitoreo industrial, Smart Home (control de: calefacción, ventilación, aire acondicionado, iluminación, acceso, detención de incendios y consumo energético), Smart Health, Smart City, seguimiento de activos y refrigeración, entre otros (Vasseur, 2014).

4.1 Características

Los dispositivos que conforman las redes LLN de tipo IEEE 802.15.4 presentan las siguientes características (Espinosa Gualotuña, 2015):

- Rango de transmisión de datos a 868 MHz: 20kbps; 915 MHz: 40 kbps y 2,4 GHz: 250kbps.
- Alcance de 10 a 200 m.
- Latencia menor a los 15mseg.
- Canales: 868/915 MHz: 11canales y 2.4GHz:16 canales.
- Bandas de frecuencia 868/915 MHz y 2.4 GHz.
- Direccionamiento corto de 16 bits o 64 bits IEEE.

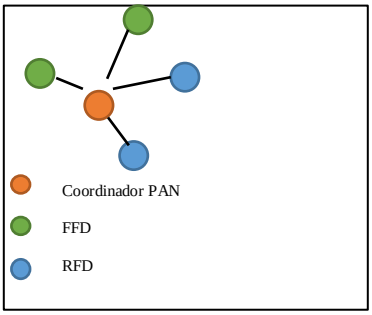
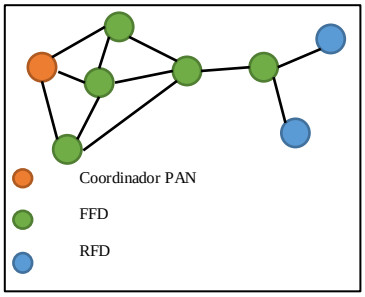
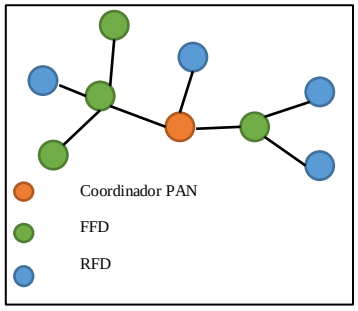
- Canal de acceso CSMA – CA y CSMA – CA ranurado.
- Rango de temperatura industrial -40 a +85°C.
- Permite alternativamente usar una topología en estrella o peer to peer.
- Posibilita la reserva de slots de tiempo garantizado, GTS.
- Define un protocolo con confirmación para asegurar la fiabilidad de las comunicaciones.
- Se orienta a la obtención de un bajo consumo.
- Proporciona un sistema de detección de energía ED.
- Posee un mecanismo de indicación de la calidad del enlace, LQI.

Además, teniendo en cuenta su funcionalidad el estándar IEEE 802.15.4 especifica dos tipos de dispositivos: los que poseen una funcionalidad completa o dispositivos de funciones plenas FFD (Full function Devices) y los de funcionalidad reducida o RFD (Reduced Function Devices) (Menchaca Paz, 2012).

Los FFDs al tener implementadas todas las funciones de los dispositivos pueden ejercer como coordinadores de una red PAN y desempeñarse como nodos intermedios para retransmitir la información de sus nodos vecinos a su destino. Los RFD son dispositivos de menor capacidad, generalmente son sensores o actuadores que optimizan su trabajo, gestionando la información retransmitida y permitiendo que su destino se encargue de procesar los datos, evitando hacerlo localmente para reducir el consumo de energía y alargar la vida de su batería (Vera Pérez, 2017). Estos sistemas cuentan con topologías de red cuya descripción se observa en la tabla 1.

Tabla 1.

Topologías de red IEEE 802.15.4

Topología	Descripción
Estrella  *Figura 1.* Topología estrella *Adaptación (Menchaca Paz, 2012)*	Determina un dispositivo FFD como coordinador de la PAN o nodo central y la comunicación se establece entre cada dispositivo y el coordinador de la red.
Peer – To -Peer  *Figura 2.* Topología peer- to- peer *Adaptación* (Menchaca Paz, 2012)	También hay un coordinador, pero los dispositivos pueden comunicarse con cualquier otro dispositivo que este a su alcance. Sin embargo, solo los FFDs tienen la capacidad para elegir el camino más adecuado.
Clúster- Tree  *Figura 3.* Topología clúster-tree *Adaptación* (Menchaca Paz, 2012)	Posee un coordinador de red global que gestiona la red al que se conectan otros nodos, algunos de estos nodos pueden ser coordinadores locales gestionando otros nodos simples.

Para entender el funcionamiento de las redes LLN se debe hablar de la arquitectura del protocolo IEEE 802.15.4, la cual se especifica en capas como se observa en la figura 4, cada una responsable de una parte del estándar y de ofrecer servicios a las capas superiores. En los párrafos siguientes se describe cada una de las capas.

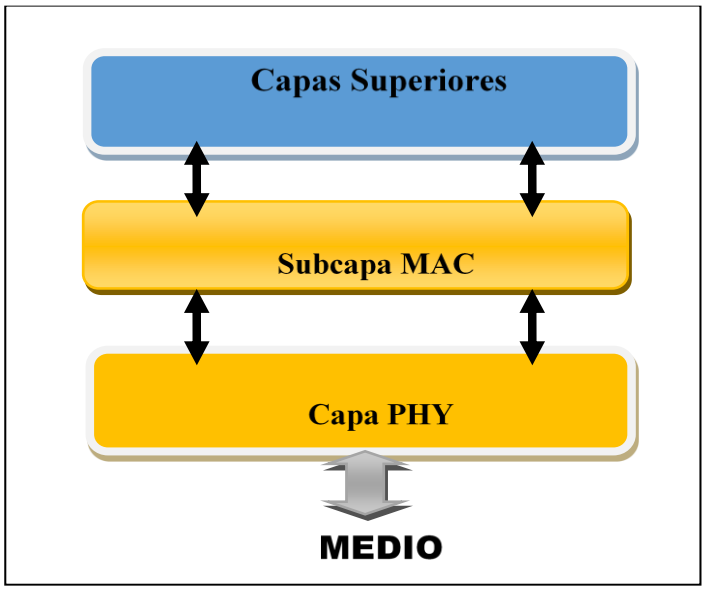


Figura 4. Arquitectura de estándar IEEE 802.15.4
Adaptación a partir de la figura 5.4 de (IEEE Std 802.15.4-2015, 2016)

4.2 Capa física (PHY)

Es la responsable de propagar la información por el medio de transmisión, estableciendo las propiedades físicas y eléctricas de los componentes del hardware (San José, 2015). Las características del PHY son la activación y desactivación del transceptor de radio, la ED (Energy Detection), la LQI (Link Quality Indicator), la selección de canales, la verificación de que el canal esté desocupado por medio del CCA (Clear channel assessment), el rango y la transmisión, así como la transmisión y la recepción de paquetes a través del medio físico (IEEE Std 802.15.4-2015, 2016).

La capa PHY admite diferentes bandas de frecuencias, así como diferentes tipos de modulaciones, además de otras características como velocidad de transmisión y número de canales, como se ilustra en la tabla 2. (Vera Pérez, 2017)

Tabla 2.

Características de canales IEEE 802.15.4

Banda	Canales	Modulación	Ancho de Banda	Región
868 MHz	1	BPSK	20 kbps	Europa, Japón
915 MHz	10	BPSK	40 kbps	EE. UU.
2.4 GHz	16	O-QPSK	250 kbps	Internacional

Nota: Adaptación a partir de la tabla 2.1 de (Vadillo Gutiérrez, 2014)

Cabe señalar que todas estas frecuencias emplean una técnica denominada DSSS (Direct Sequence Spread Spectrum), que generalmente se utiliza para modular digitalmente una portadora, aumentando el ancho de banda de la transmisión y disminuyendo la densidad de potencia espectral. Es decir, la señal resultante tiene un espectro muy parecido al del ruido, de tal manera que a todos los receptores les parecerá ruido, menos al que va dirigida la señal quien lo puede decodificar porque comparte la misma secuencia (Vera Pérez, 2017).

En cuanto a la PHY de 2.4 GHz es una banda libre a nivel mundial, operando particularmente en el área industrial, médica y científica; mientras que la PHY de los 868/915 MHz define operaciones en la banda de 865 MHz en Europa y 915 MHz en la banda ISM (Industrial, Scientific and Medical) en Estados Unidos (Espinosa Gualotuña, 2015).

4.3 Subcapa MAC

Esta subcapa tiene como función principal controlar el acceso al canal de radio a través de diferentes tipos de mecanismos, aparte de suministrar dos servicios: el servicio de datos MAC y la administración de servicios MAC que se interconectan al punto de acceso de servicio (SAP) de la entidad de administración de subcapa MAC (MLME) (conocido como MLME-SAP) (IEEE Std 802.15.4-2015, 2016).

El servicio de datos MAC permite la transmisión y recepción de MPDU (MAC Protocol Data Unit) a través del servicio de datos de la capa física. Así mismo, el servicio de gestión MAC se encarga de los mecanismos para la gestión y mantenimiento de la base de datos de objetos de la subcapa MAC (San José, 2015).

Esta capa presenta características tales como: la administración de las beacons, el acceso al canal, la GTS (gestión de ranuras de tiempo garantizadas), validación de tramas, retransmisión de mensajes de verificación, asociación y disociación. Adicionalmente, la subcapa MAC proporciona herramientas para implementar mecanismos de seguridad apropiados para la aplicación (IEEE Std 802.15.4-2015, 2016).

4.4 Seguridad

Todos los sistemas informáticos deben cumplir con ciertos objetivos de seguridad de la información como la disponibilidad, la confidencialidad y la integridad de los datos. Por consiguiente, las redes LLN deben asegurar que se cumplan estos criterios, pero al igual que otras redes inalámbricas presentan varias vulnerabilidades. Siendo una de ellas el acceso al

medio inalámbrico el cual está abierto para todos, donde un adversario hostil podría acceder a la red sin estar siquiera cerca, capturando la información y alterando las transmisiones de la red. Además, una red LLN puede estar compuesta por cientos o miles de nodos de sensores que ingresan y salen de la red en cualquier instante, dificultando la seguridad en una red tan dinámica. Otra debilidad es la limitación de las redes LLNs en cuanto a energía, almacenamiento y procesamiento de cómputo, la cual restringe la elección de técnicas criptográficas que se puedan aplicar para garantizar la privacidad e integridad de los datos. Finalmente, existe la posibilidad de que los nodos sean capturados, dañados o destruidos debido a la manipulación por acceso físico de un agente malintencionado (Amin & Abdel-Hamid, 2016).

Por otro lado, en el estándar IEEE 802.15.4 se definen los mecanismos criptográficos basados en clave simétrica (la misma clave para cifrar y descifrar), los cuales se apoyan en procesos de las capas superiores, para el manejo y la gestión de las claves. Así mismo, el mecanismo criptográfico suministra combinaciones particulares de los siguientes servicios de seguridad: Confidencialidad de los datos (la información transmitida solo se difunde a los interesados), Autenticidad de los datos (la información transmitida por la fuente no se modifique en el proceso), Protección de reproducción (la información duplicada sea detectada) (IEEE Std 802.15.4-2015, 2016).

Adicionalmente, para minimizar la sobrecarga de seguridad en las tramas transmitidas, el estándar especifica que el cifrado se realice trama a trama, habilitando diferentes niveles de autenticidad y confidencialidad de los datos. Pero, cuando se requiere protección relevante, siempre se proporciona protección de reproducción (garantizando que la información duplicada sea detectada) (IEEE Std 802.15.4-2015, 2016).

La protección de trama criptográfica utiliza una clave compartida en un par de dispositivos o dentro de un grupo de dispositivos denominados: clave de enlace y clave de grupo respectivamente, de este modo se permite alguna flexibilidad entre el mantenimiento y costo de claves frente a la protección criptográfica proporcionada. Si una clave de grupo es usada para una comunicación punto a punto, esta protección provee seguridad únicamente contra dispositivos externos y no contra dispositivos hostiles dentro del grupo de claves compartidas. (IEEE Std 802.15.4-2015, 2016)

Generalmente los ataques se consideran provenientes de la parte interna de la red donde, el nodo intruso tiene acceso lógico a la red, esto se puede evitar impidiendo que un nodo intruso se asocie a la red establecida; es poco probable considerar que un ataque provenga desde afuera de la red (Diaz Suárez, 2017). De ahí que el estándar IEEE 802.15.4 fije aspectos fundamentales para asegurar la autenticación de los nodos de comunicación evitando que cualquier nodo intruso la vulnere. Así mismo, el estándar especifica el tipo de algoritmo de cifrado que debe emplearse para las operaciones de criptografía (Diaz Suárez, 2017).

El IEEE 802.15.4 establece como algoritmo de cifrado al estándar AES (Advanced Encryption Standard), con cifrado simétrico, el cual posee una longitud de clave de 128 bits, este algoritmo es utilizado tanto para cifrar como para validar la información; para realizar este procedimiento utiliza un MIC (Message Integrity Code) o MAC (Message Authentication Code) añadido al final del mensaje, este código asegura la integridad de la información que se está transmitiendo (cabecera MAC y datos o payload), simultáneamente ayuda a garantizar que el emisor sea quien dice ser. Por tanto, al recibir una trama de un nodo que no es confiable el código de integridad no debe coincidir porque se generó con una clave diferente. (Diaz Suárez, 2017) (García Arano, 2010).

5. Internet de las cosas en redes LLN

El término Internet de las cosas se escuchó por primera vez en 1999, cuando el británico Kevin Ashton lo utilizó para exponer un sistema que consistía en conectar a Internet las etiquetas de identificación por radiofrecuencias RFID (Radio Frequency Identification) y sensores, utilizados en cadenas de supermercados para contar y rastrear los productos sin intervención humana (Rose, Karen; Eldridge, Scott; Chapin, 2015).

Teniendo en cuenta las definiciones de IoT en (Minerva, Biru, & Rotondi, 2015) por el IEEE, el ITU y la IETF, se podría sintetizar el concepto de Internet de las Cosas como la interconexión entre objetos y personas a través de distintas redes de comunicación para compartir información y tomar decisiones de forma independiente. Sin embargo, a pesar de los numerosos y diversos puntos de vista, aún no existe una definición exclusiva y universalmente aceptada para IoT.

Figura 5. Áreas de aplicación para Internet de las cosas
Adaptado de (Vergara & Villan, 2017)

Así mismo, IoT pretende facilitar la forma como nos relacionamos con el entorno, como se aprecia en la figura 5, busca crear soluciones, implementando aplicaciones en áreas como la seguridad, el transporte, la salud, los servicios públicos, la industria, la energía, la agricultura, la automatización del hogar, el medio ambiente entre otros; apoyándose en diferentes tecnologías como: WiFi, 4GLTE, Weightless, EnOcean , Bluetooth, Z-Wave, WiFi de baja potencia, Zigbee, Sigfox, LoraWAN, WirelessHART, NFC, RFID y WSN. Además, los sistemas y aplicaciones de la IoT están diseñados para proporcionar seguridad, privacidad, protección, integridad, confianza, fiabilidad, transparencia, anonimato, y están sujetos a limitaciones éticas (Liñán, Vives, Bagula, Zennaro, & Pietrosemoli, 2015).

En algunos escenarios IoT requiere tecnologías como las redes de baja potencia y con pérdidas que se ajustan a las necesidades de ciertos casos de uso, donde la comunicación, el control y la monitorización de dispositivos son necesarios con una transmisión de datos a baja velocidad y con un bajo consumo de energía, como no lo hacen otras tecnologías.

La tecnología LLN funciona de forma autónoma sin intervención humana como redes ad hoc auto configurables; están compuesta por microcontroladores energéticamente eficientes, transceptores de radio de baja potencia y sensores inalámbricos, estos últimos interconectados entre sí y alimentados por baterías; encargados de capturar la información del medio, útiles en ambientes donde se busca controlar las condiciones del entorno para dotarlo con más y mejores servicios. Dichas características plantean desafíos, por un lado, en términos de software que se ejecuta en objetos inteligentes, y por otro lado en términos de protocolos de red, que los objetos inteligentes utilizan para comunicarse (Zikria, Yu, Afzal, Rehmani, & Hahm, 2018).

5.1 Arquitectura de IoT, basada en IETF, IEEE 802.15.4 y 6LoWPAN

Se han propuesto diferentes arquitecturas para IoT, sin embargo, aún no se confluye a un modelo de referencia, la mayoría de los investigadores han orientado sus modelos de arquitecturas en diferentes aspectos de IoT.

Para este caso en particular, se toma como referencia una arquitectura de 5 niveles, ilustrada en la Figura 2, con base en modelos conocidos como OSI (Open System Interconnection) y TCP/IP; esta arquitectura se elige porque de forma conceptual describe completamente la lógica de la tecnología IoT sobre IEEE 802.15.4 y usando TCP/IP.

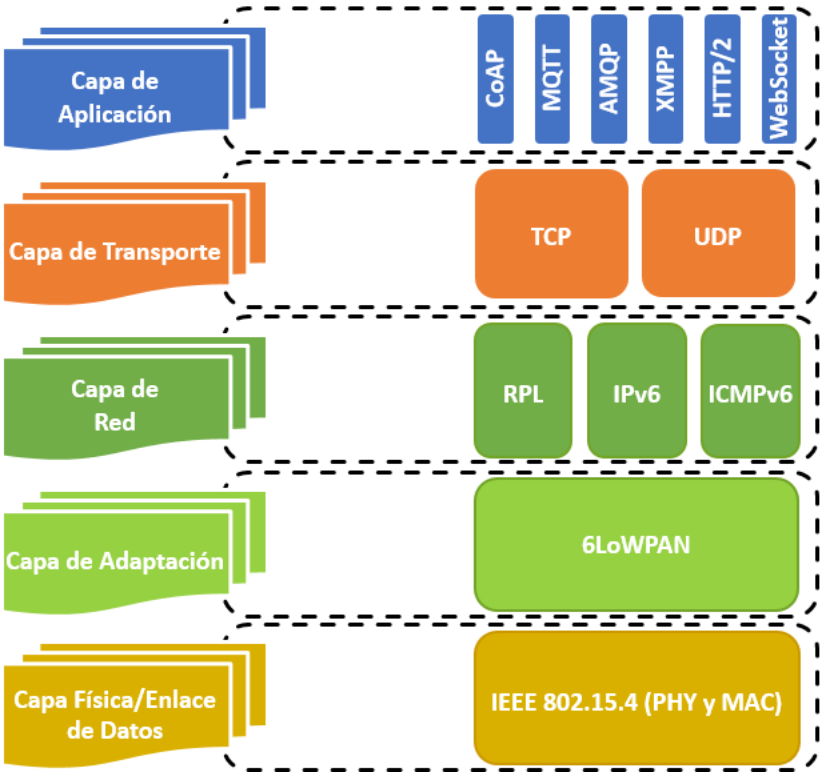


Figura 6. Arquitectura TCP/IPv6 para IoT en IETF para redes IEEE 802.15.4

Adaptación a partir de figura 7.2 de (Tsiatsis, Karnouskos, Höller, Boyle, & Mulligan, 2018)

1. **Capa Física/Enlace de Datos.** Esta es una capa de hardware, la cual está formada por diferentes dispositivos de bajo costo como tarjetas inteligentes, RFID y redes de sensores, que tienen como función recopilar información de los objetos (temperatura, humedad, localización, etc.) para posteriormente convertirla en señales digitales y poder transmitirlas por la red (Cobos, 2016). Dentro de los estándares más usados según (Naik, 2017) (Gardašević et al., 2017) se encuentran: el protocolo IEEE 802.15.4 (LR-WPAN) o redes inalámbricas de área personal, Bluetooth de baja energía, Wifi, NFC (comunicación de campo cercano) y 3GPP (Proyecto Asociación de Tercera Generación).

Además, esta capa se encarga del manejo y tratamiento de los datos tomados por la capa física. En esta capa se conforman los mensajes, se encarga de los errores de transmisión y proporciona una interfaz bien definida para la capa de red. El estándar IEEE 802.15.4 define los requisitos en esta capa y tiene como función gestionar el acceso al canal y realizar el mantenimiento de la PAN (Red de área personal).

Cabe señalar la evolución que ha tenido el estándar IEEE 802.15.4, en su primera versión lanzada en 2003, soportaba dos capas físicas. Más tarde, hubo otra revisión lanzada en 2006, que mejoró las velocidades de transferencia. y se agregaron bandas adicionales en las revisiones posteriores (Devadiga, 2011). En 2011, el estándar incluye las tres enmiendas aprobadas después de la revisión de 2006, agregando cuatro opciones PHY más, junto con la capacidad MAC para admitir el rango (IEEE Std 802.15.4-2015, 2016).

La revisión actual del estándar es la aprobada en el año 2015, que agrupa todas las correcciones realizadas desde el año 2011, incluyendo la enmienda IEEE 802.15.4e del 2012, que incluye diferentes modos de operación con el objetivo de dar soporte y cobertura a mercados industriales, y la enmienda IEEE 802.15.4g, que incluye tres nuevas PHY para ampliar el

alcance de la comunicación y aumentar la confiabilidad para cumplir con los requisitos de las redes inalámbricas de servicios inteligentes Wi-SUN (Wireless Smart Metering Utility Network) (Vera Pérez, 2017) (San José, 2015).

Tabla 3.

Evolución del Estándar IEEE 802.14.5

Estándar	Año	Especificaciones
IEEE802.15.4	2003	Estableció dos PHY (capa física) opcionales, que operan en diferentes bandas de frecuencia (868 MHz, 915 MHz y 2.4 GHz.) con un MAC (Control de Acceso Medio) sencillo y efectivo (Vera Pérez, 2017). Para filtrar interferencias en las transmisiones, utiliza el mecanismo de ensanchado de espectro DSSS (Direct Sequence Spread Spectrum) (Alcázar, 2013).
IEEE802.15.4	2006	Según (IEEE Computer Society, 2015), agregó dos opciones más a la PHY. El MAC continúa siendo compatible con versiones anteriores, pero la revisión agregó tramas MAC con un número de versión aumentado y una variedad de mejoras MAC, que incluyen lo siguiente: • Soporte para una base de tiempo compartida con un mecanismo de sellado de tiempo de datos. • Soporte para la programación de balizas. • Sincronización de mensajes de difusión en redes de área personal habilitadas para balizas PAN. • Mejora de la seguridad de la capa MAC.
IEEE802.15.4	2011	Según (IEEE Computer Society, 2015), la norma fue revisada para incluir las tres enmiendas aprobadas después de la revisión de 2006. Este esfuerzo agregó cuatro opciones PHY, junto con la capacidad MAC para admitir el rango. Además, se cambió la organización del estándar para que cada PHY tuviera una cláusula separada, y la cláusula MAC se dividió en descripción funcional, especificación de interfaz y especificación de seguridad.
IEEE 802.15.4e	2012	Según (Alcázar, 2013), el estándar incorpora nuevos modos de operación tales como: • DSME (Deterministic & Synchronous Multi-channel Extension) • LLDN (Low Latency Deterministic Network) • TSCH (Time Slotted Channel Hopping) • RFID Blink (Radio Frequency Identification Blink) • AMCA (Asynchronous Multi-Channel Adaption)

		IEEE 802.15.4e también ha desarrollado funciones MAC adicionales para mejorar las capacidades generales, como baja energía (LE), elemento de información (IE), balizas mejoradas (EB) y solicitudes de balizas mejoradas (EBR), etc (Sum, Zhou, Kojima, & Harada, 2017). En (San José, 2015), el estándar IEEE 802.15.4e agrega dos mecanismos que garantizan un consumo menor de energía: • LE (Low Energy) • CSL (Coordinated Sampled Listening) • RIT (Receiver Initiated Transmissions)
IEEE 802.15.4g	2012	Define tres capas físicas alternativas PHY que sirven para proporcionar eficiencia energética y velocidad en los datos. Sus principales características son: • FSK (Modulación por desplazamiento de frecuencia) provee velocidades de datos entre 5 y 400 Kbps, según los parámetros de radio, y utiliza la FEC (corrección de errores hacia adelante). • OQPSK (Codificación de desplazamiento de fase en cuadratura compensada) provee velocidades de datos en el rango de 6 a 500 Kbps. • OFDM (Multiplexación por división de frecuencia ortogonal) provee altas velocidades de datos, en el rango de 50–800 Kbps, en entornos desafiantes con condiciones de desvanecimiento de múltiples rutas. Para transportar un datagrama IP completo en una sola trama de datos, sin la necesidad de fragmentarlo, el estándar IEEE 802.15.4g extiende la longitud máxima del paquete a 2047 bytes y, en consecuencia, adopta un FCS de 4 bytes. (Sum et al., 2017)
IEEE 802.15.4	2015	Según (IEEE Computer Society, 2015), las características agregadas por las enmiendas incluyen lo siguiente: • Formatos de cuadro mejorados que mantienen la compatibilidad con versiones anteriores. • Elementos de información (IE) • Agilidad de canal • Opciones de supertrama extendidas • Mecanismos de baja energía. • Un marco de reconocimiento mejorado que puede transportar datos y puede asegurarse. • Acceso prioritario al canal. • Una variedad de nuevas opciones de modulación, codificación y banda PHY para admitir una amplia variedad de necesidades de aplicaciones, incluida la identificación por radiofrecuencia (RFID), redes de servicios inteligentes (SUN), operación de espacios en blanco de televisión (TVWS), monitoreo de infraestructura crítica de bajo consumo de energía (LECIM), y comunicaciones y control ferroviario (RCC).

2. Capa de adaptación. La capa de adaptación proporciona los mecanismos de fragmentación y reensamblaje de los paquetes IPv6, así como la compresión de las cabeceras. Cuando los paquetes IPv6 no pueden ajustarse dentro de los 102 bytes de tamaño de la trama de carga útil de la MAC, los paquetes son fragmentados en múltiples tramas de la capa de enlace para acomodarse al mínimo MTU de IPv6 requerido, para ser re-ensamblados en el otro extremo (Harz., Franco, & Pinto, 2012).

Para fragmentar y reensamblar estas tramas se usa el protocolo 6LowPAN de estándar abierto, definido por la IETF el cual, a través de la encapsulación y compresión de cabeceras, transmite y recibe paquetes IPv6 sobre enlaces IEEE 802.15.4; su uso se especifica en los documentos RFC 4919 y RFC 6282 que proponen una capa intermedia de adaptación que permite a los dispositivos limitados soportar el protocolo IP.

Específicamente, 6LoWPAN en la capa de adaptación realiza las siguientes tres optimizaciones para reducir la sobrecarga de comunicación:

- Compresión de encabezado: 6LoWPAN comprime el encabezado de los paquetes IPv6 para disminuir la sobrecarga de IPv6, como en las figuras 7 y 8. Para ellos se eliminan algunos campos que no son estrictamente necesarios (Sethi & Sarangi, 2017).

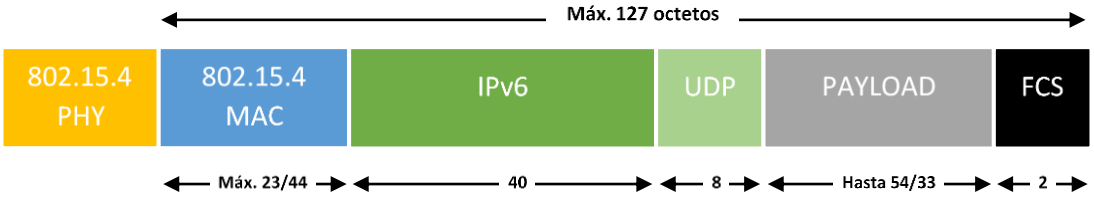


Figura 7. Trama 6LoWPAN sin compresión de cabecera IPv6

Adaptación de la figura 3-17 (Menchaca Paz, 2012)

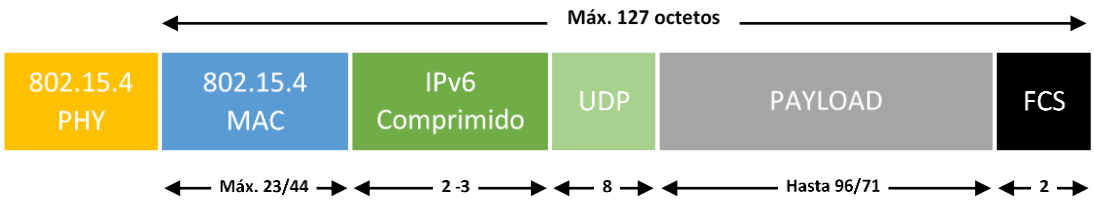


Figura 8. Trama 6LoWPAN con compresión de cabecera IPv6

Adaptación de la figura 3-17 (Menchaca Paz, 2012)

- Fragmentación: El protocolo 6LoWPAN en la capa de adaptación propone fragmentar el paquete IPv6 con una MTU de 1280 bytes para poderlo transmitir en la trama del estándar IEEE 802.15.4 con MTU de 127 bytes (Sethi & Sarangi, 2017).

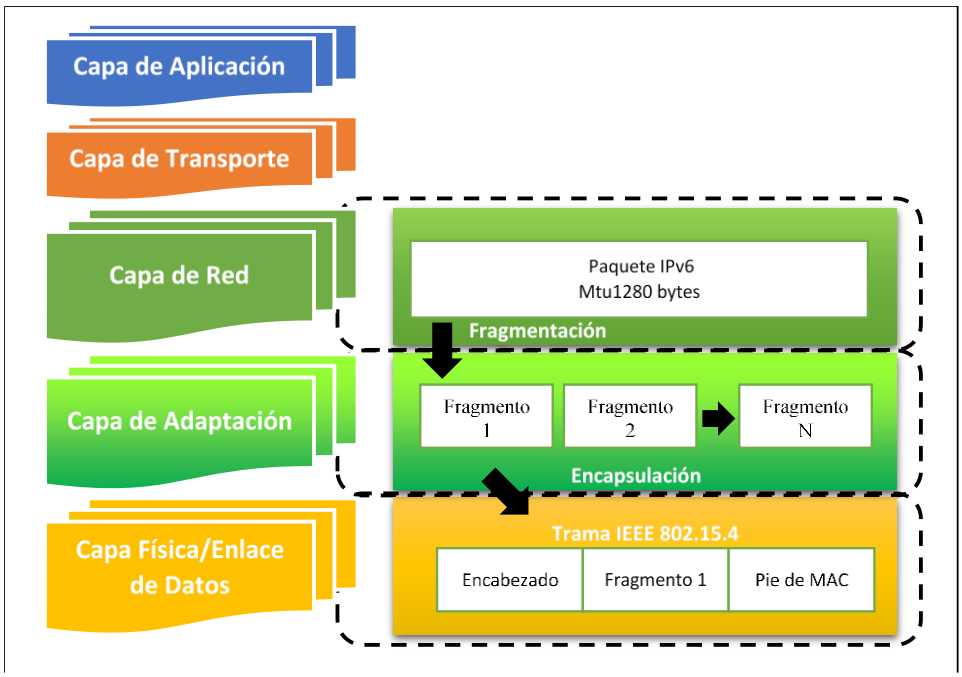


Figura 9. Proceso de fragmentación de un paquete IPv6 sobre IEEE 802.15.4

Adaptación a partir de la figura 6 de (Carrillo, 2017)

- Enrutamiento: 6LoWPAN a través de su capa de adaptación puede enviar paquetes de un dispositivo a otro mediante múltiples saltos y puede soportar el enrutamiento en la capa de

enlace por medio del mesh-under y en la capa de red por medio del route-over (Carrillo, 2017).

3. Capa de red. La capa de red transmite y procesa la información obtenida de la capa de enlace de datos. Se encarga de enrutar los datos (maneja la transferencia de paquetes del origen al destino) y de encapsular el datagrama IPv6 en las tramas de la capa de enlace de datos para implementar aplicaciones de IoT. Los estándares utilizados para el enrutamiento de los datos de la capa de red IoT incluyen el protocolo de enrutamiento RPL, mientras que para la encapsulación el estándar usado es IPv6 (Sharma & Gondhi, 2018). A continuación, se describen brevemente los protocolos de red RPL, IPv6 e ICPMv6 respectivamente:

- El Protocolo RPL

RPL (Routing Protocol for Low-Power and Lossy Networks) es un protocolo de enrutamiento dinámico de tipo vector distancia, desarrollado por el grupo ROLL (Routing Over Low-Power and Lossy Networks) de la IETF para las redes LLNs que utilizan IPv6, definido en el documento RFC 6550. El objetivo de RPL es suministrar eficientes caminos de enrutamiento para diferentes tipos de tráfico: punto a punto, punto a multipunto y multipunto a punto (Harz. et al., 2012). Para ello, tiene definida una OF (Función Objetivo) compuesta por una combinación de métricas y restricciones que permiten elegir la mejor ruta (Morillo, 2017).

La topología de RPL es similar a un árbol, representada por un grafo o DAG (Direct Acyclic Graph), donde todos los enlaces están orientados para que no se produzcan ciclos (Vera Pérez, 2017). Este DAG está compuesto por uno o varios nodos raíces (root nodes) que actúan como nodos sumideros, y partiendo de estos nodos, el grafo DAG se va dividiendo en nodos hijos que

se comunican con los nodos raíz a través de mensajes DODAG (Destination-Oriented DAG), característico de cada nodo raíz (Morillo, 2017).

Para la construcción de un DODAG se escogen los nodos establecidos como nodos DODAG roots que contienen la configuración del DODAG; estos nodos enviarán mensajes DIO (DODAG Information Object) de tipo broadcast a sus vecinos con información relevante. El resto de los nodos escuchara los mensajes DIO y utilizaran esta información para unirse al nuevo DODAG y escoger al mejor padre o seguir perteneciendo al mismo DODAG, estas decisiones dependen de cómo se haya definido su Función Objetivo, su Rank (posición de un nodo con respecto a la posición del nodo DODAG root) y los de sus vecinos (Morillo, 2017). A medida que los nodos reciben los mensajes DIO de sus vecinos, se construyen diferentes entradas en una tabla de encaminamiento para cada dirección de los mensajes DIO, y después de escoger cuál será su padre, el nodo calcula el valor de su Rank para volver a propagar el mensaje DIO con el fin de que los nodos que están debajo realicen el mismo proceso (Vera Pérez, 2017). Este procedimiento se representa en la figura 10.

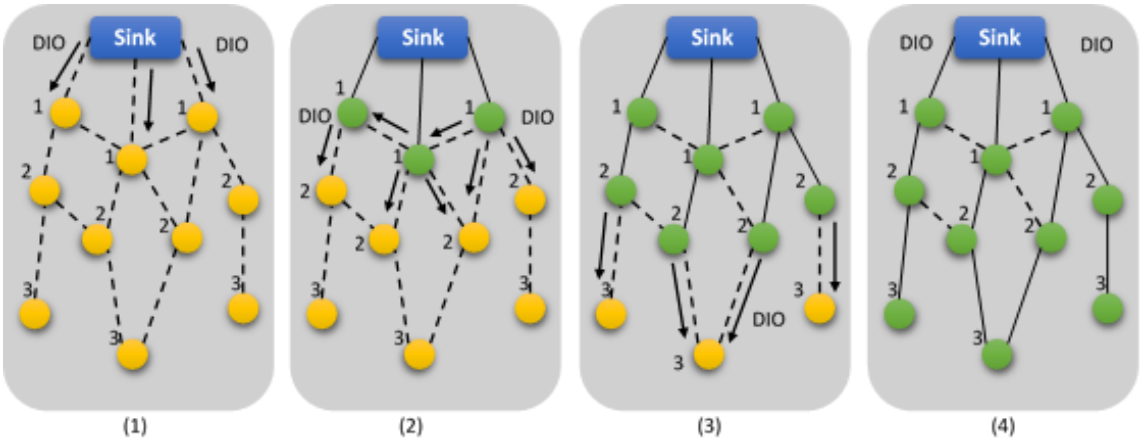


Figura 10. Formación del DODAG en RPL

Adaptación de la figura 2 (Vera Pérez, 2017)

Además, para la formación y el mantenimiento del DODAG, RPL utiliza mensajes de control definidos en paquetes ICMPv6, como en la figura 11, los cuales se describen a continuación (Fàbregas Bachs, 2016):

- ***DIS*** (solicitud de información del DODAG): utilizado para solicitar información de DODAG cercanos, utilizados para descubrir redes existentes.
- ***DIO*** (objeto de información del DAG): mensaje que comparte información del DAG, enviado como respuesta a mensajes DIS, utilizado periódicamente para refrescar la información de los nodos sobre la topología de la red.
- ***DAO*** (objeto de actualización al destino): enviado en dirección hacia el DODAG, es un mensaje enviado por los equipos para actualizar la información de sus nodos

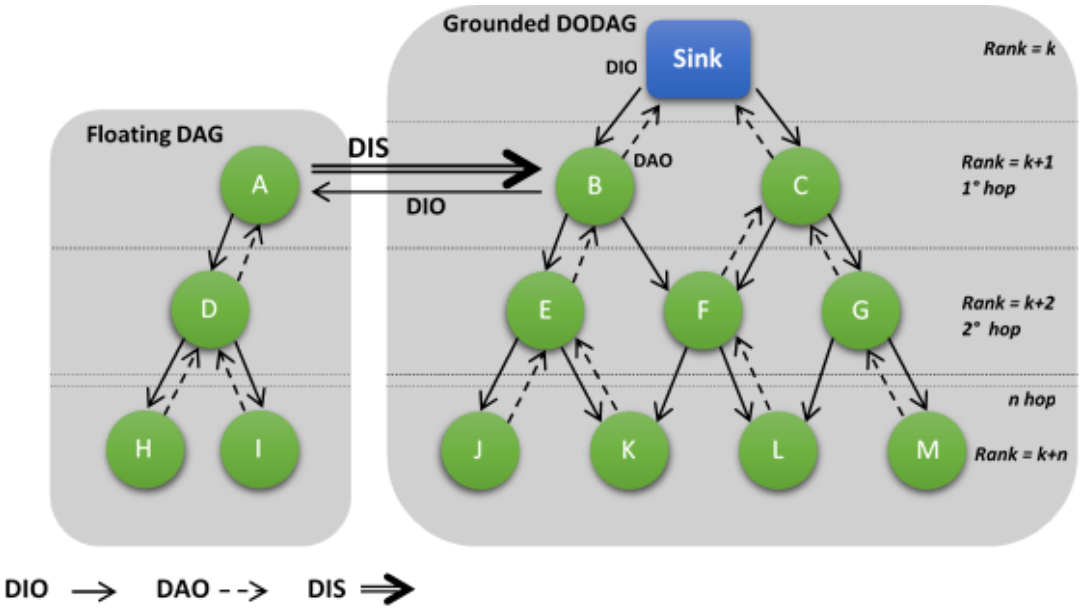


Figura 11. Mensajes ICMPv6 RPL: DIO, DAO y DIS
Adaptación de la figura 15 (Fàbregas Bachs, 2016)

- Protocolo IPv6 (Internet Protocol version 6)

El protocolo IP versión 6 es una nueva versión del protocolo IP (Internet Protocol), para reemplazar en forma gradual a IPv4, definida en el documento RFC 8200. El motivo para desarrollar este nuevo protocolo es el agotamiento de direcciones de red necesarias en Internet (Vadillo Gutiérrez, 2014). Por ello, IPv6 utiliza un formato de dirección de 128 bits, lo que permite 2^{128} o aproximadamente $3.4\,x10^{38}$direcciones, equivalente a $8x10^{28}$ veces más que IPv4. En la tabla 4 se observan algunas características de IPv6.

Tabla 4.

Características IPv6

Parámetro	IPv6
Estandarizado por	IETF
Interoperabilidad	No
Función	Proporciona un sistema de identificación y ubicación para computadores en redes y enruta el tráfico a través de Internet.
Diseñado para/Usado para	Internetworking de paquetes conmutados
Tipo de Protocolo	Protocolo de capa de Internet
Enrutamiento	Enrutamiento de multidifusión eficiente simplificado
Aplicación	Proporciona transmisión de datagramas de extremo a extremo a través de múltiples redes IP
Seguridad	IPsec y soporte integrado de autenticación y privacidad
Escalabilidad	Esquema de direcciones altamente escalable

Nota: Adaptado de la tabla II (Sharma & Gondhi, 2018)

- Protocolo ICMPv6 (Internet Control Message Protocol version 6).

ICMPv6 es un protocolo de control de información en Internet, implementado para ser utilizado en la arquitectura del protocolo IPv6 (Vadillo Gutiérrez, 2014). En la RFC 4443, DTLS es utilizado en nodos IPv6 para informar errores encontrados durante la transmisión de paquetes

entre nodos y otras funciones de la capa de Internet como por ejemplo el diagnóstico de la red, con el uso de ping ICMPv6. En la tabla 5 se muestran las principales características del protocolo ICMPv6.

Tabla 5.

Características ICMPv6

Parámetro	ICMPv6
Estandarizado por	Network Working Group.
Interoperabilidad	No
Tipo de Protocolo	Protocolo de capa de red.
Función	• Detección e informe de errores encontrados en la interpretación de paquetes IPv6. • Realiza diagnósticos de red. • Descubrimiento de nodos vecinos. • Implementa Multidifusión.
Tipo de mensajes que emplea	MENSAJES DE ERROR • Destino Inalcanzable • Paquetes demasiado grandes • Tiempo excedido • Problema de parámetros MENSAJES DE INFORMACIÓN • Petición Eco • Respuesta Eco
Estructura de los mensajes	El formato genérico de un paquete está compuesto por tres campos: • Tipo: Indica si es un mensaje de error o informativo. • Código: Informa el tipo de mensaje ICMPv6. • Checksum: Detecta errores en los mensajes ICMPv6.

4. Capa de transporte. Esta capa se ocupa del transporte de la información desde la fuente al destino, a través de puertos (Sockets). Para ello, los mensajes de nivel de aplicación se

distribuyen en pequeñas unidades (Segmentos) y posteriormente se envían a la red (González García, 2017). En la capa de transporte se realiza la detección y corrección de errores de datos, el control del flujo y la secuenciación, para garantizar que todos los segmentos lleguen de forma correcta a su destino. Dentro de la arquitectura TCP/IP las funciones de la capa de transporte se obtienen con TCP (Transmission Control Protocol), protocolo de transporte dominante en Internet.

No obstante, el protocolo TCP no resulta óptimo para las comunicaciones IoT de tipo IEEE 802.15.4, por lo que al establecer la conexión con el protocolo TCP se genera una sobrecarga al transmitir pequeñas cantidades de datos, lo cual se traduce en un mayor consumo de energía.

Adicionalmente, la característica del protocolo TCP de entrega efectiva de paquetes, puede introducir retardos que impiden que dispositivos como los actuadores no realicen de forma eficiente su función. Además, el tamaño de la cabecera es una limitante para redes de tipo 802.15.4 donde el MTU a nivel de MAC es de 127 bytes. Para lo cual UDP (User Datagram Protocol) es más conveniente debido a su tamaño de cabecera e implementación simple.

El protocolo UDP cuenta con la ventaja de proveer la entrega de datos sin utilizar muchos recursos, sin embargo, no proporciona ninguna garantía de entrega y no hay protección contra la duplicación de datagramas, pero su simplicidad reduce la sobrecarga del protocolo y puede ser adecuado para algunas aplicaciones en lugar de TCP.

El envío de paquetes UDP en un dispositivo, se realiza sin necesidad de establecer una conexión previa con el destinatario, además envía paquetes sin control de flujo, es decir los paquetes no llegan en el mismo orden que se transmitieron y tampoco hay confirmación de entrega por lo que no se sabe si llegaron correctamente, generalmente este protocolo es

apropiado en aplicaciones donde la velocidad es más crítica que la confiabilidad (González García, 2017).

5. **Capa de aplicación**. Es la responsable de entregar servicios específicos al usuario, dar formato y presentación de los datos y tomar todas las decisiones de control, seguridad y administración de aplicaciones (Gonzalez, Flauzac, & Nolot, 2018).

Las aplicaciones IoT se pueden implementar para diferentes tipos de servicios; por ejemplo, en un hogar inteligente el usuario puede solicitar parámetros específicos tales como mediciones de calefacción, ventilación y aire acondicionado o valores de temperatura y humedad. Así mismo, las implementaciones abarcan otras áreas como: domótica, ciudades inteligentes, logística, comercio, medio ambiente, seguridad pública, salud, entre otras.

Los protocolos más utilizados en esta capa son: CoAP (Constrained Application Layer Protocol) se usa para transferencia de documentos web en entornos restringidos, MQTT (Message Queuing Telemetry Transport Protocol) es un protocolo de transporte de mensajes enfocado a la conectividad M2M ligeras en redes restringidas., AMQP (Advanced Message Queuing Protocol) se enfoca en el intercambio de mensajes entre distintas plataformas, XMPP (Extensible Messaging and Presence Protocol) es un estándar de mensajería instantánea, utilizado para multichat, telepresencia y llamadas de voz y video., WebSocket es un protocolo que se desarrolló para disminuir la sobrecarga de comunicación de Internet que presentaba el protocolo HTTP y HTTP/2 (Hypertext Transfer Protocol, version 2) es una actualización del HTTP/1.1, permite la transferencia de información en la web utilizando de forma más eficiente los recursos de red.

6. Protocolos de la capa de aplicación para IoT

6.1 Protocolo CoAP (Constrained Application Protocol)

CoAP es un protocolo de la capa de aplicación diseñado para la transferencia web en dispositivos inalámbricos restringidos y redes restringidas (como las definidas por el estándar IEEE 802.15.4). Este protocolo se basa en un modelo de interacción solicitud/respuesta, que se asemeja al modelo de HTTP, con la diferencia que en interacciones máquina a máquina los nodos CoAP pueden tomar roles de clientes y de servidor. En la figura 12, se ilustra el diseño de funcionamiento del protocolo CoAP. Su núcleo se especifica en RFC 7252 (Shelby, Hartke, & Bormann, 2014).

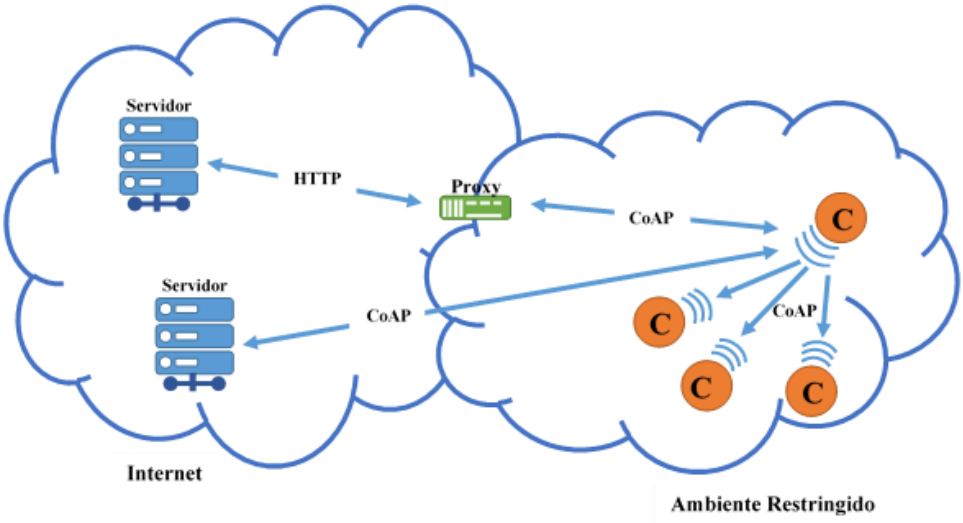


Figura 12. Diseño del protocolo CoAP
Adaptación de la figura 9 de (Castro Heredia, 2014)

Según (Shelby et al., 2014), el protocolo CoAP presenta las siguientes características:

- Cumple con los elementos fundamentales para entornos M2M restringidos.

- Permite intercambios de mensajes asincrónicos a través de enlaces UDP que admiten solicitudes unidifusión y multidifusión.
- Presenta baja sobrecarga de encabezado que reduce la complejidad de análisis.
- Posee soporte de URI y contenido (Content-Type).
- Presenta proxy simple y capacidades de almacenamiento en caché.
- Fácil de traducir a HTTP, que permite la construcción de proxies CoAP-HTTP y viceversa.
- Presenta seguridad a través de enlaces DTLS (Datagram Transport Layer Security).
- Presenta búsqueda de recursos y mecanismos de suscripción opcional (Shelby et al., 2014).

6.1.1 Estructura del protocolo CoAP (Constrained Application Protocol)

EL protocolo CoAP utiliza un enfoque de dos capas, como se observa en la figura 13. Por un lado, una capa *Mensaje*, que establece una interfaz de intercambio de mensajes con el nivel inferior de transporte UDP y, por otro lado, una capa superior *Solicitud/Respuesta*, que establece una interfaz de solicitud y respuestas con la aplicación, permitiendo la integración opcional de una capa de seguridad DTLS con la capa de transporte (Shelby et al., 2014).

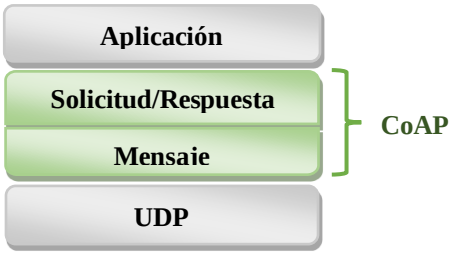


Figura 13. Resumen de capas de CoAP
Adaptado de (Shelby et al., 2014)

6.1.2 Mensaje CoAP (Constrained Application Protocol)

EL protocolo CoAP presenta dos tipos de interacciones, como son: Solicitud y Respuesta, como se observa en la figura 14. El tipo de mensaje se especifica en el campo tipo del encabezado CoAP (se define en el formato de mensaje).

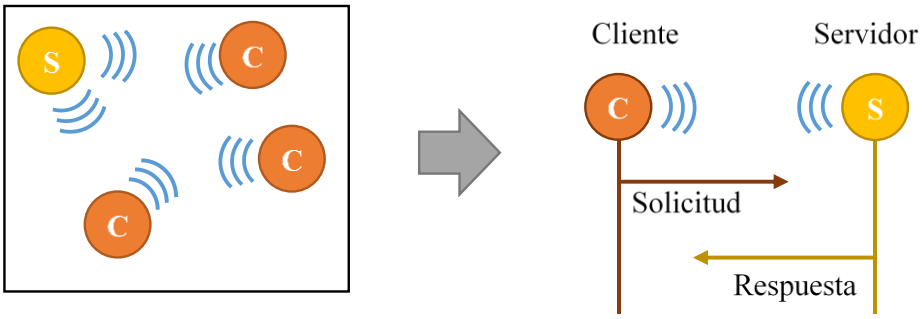


Figura 14. Interacciones del protocolo CoAP
Adaptación de (Shelby et al., 2014)

En las interacciones de los dispositivos CoAP, un cliente envía un mensaje para solicitar una acción (utilizando un código de método, GET, PUT, etc..) a determinado recurso (identificado por un URI) en un servidor. El servidor luego envía un mensaje con un código de respuesta, el cual puede incluir una representación del recurso. El protocolo CoAP define cuatro tipos de mensaje: confirmable (CON), no confirmable (NON), confirmación (ACK) y restablecimiento (RST) (Shelby et al., 2014).

6.1.3 Modelo de mensaje

El modelo de mensajes del protocolo CoAP se basa en el intercambio asíncrono de mensajes a través de enlaces UDP entre nodos finales. En (Shelby et al., 2014), se definen dos modelos de mensaje, como son: *mensajes confirmable* y *mensajes no confirmables*. En los mensajes

confirmables se envía un mensaje tipo CON (proporciona fiabilidad al mensaje) y se recibe un mensaje de tipo ACK. Los mensajes no confirmables son aquellos que se envían como mensaje tipo NON. En la figura 4 se ilustra estos dos modelos de mensaje.

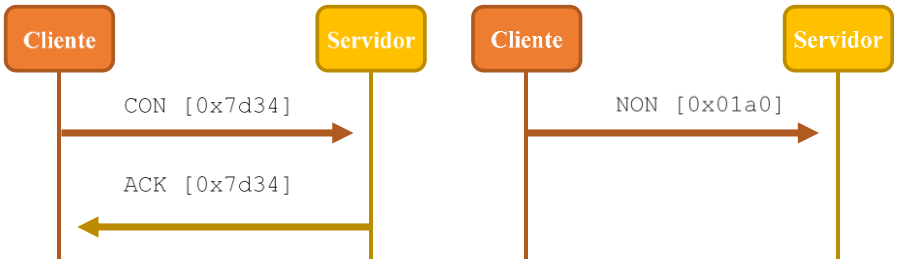


Figura 15. Mensaje confirmable y mensaje no confirmable
Adaptación de (Shelby et al., 2014)

Si el destinatario no puede procesar un mensaje confirmable o no confirmable, responde con un mensaje **RST (**en lugar de un mensaje **ACK** para el caso de un mensaje tipo CON).

Independientemente del tipo de mensaje, un mensaje puede llevar una solicitud, una respuesta o estar vacío. Un mensaje de solicitud se puede ser de tipo CON o NON, pero no como tipo ACK y RST como se ilustra en la tabla 6.

Tabla 6.

Combinaciones tipos de mensaje

	CON	NON	ACK	RST
SOLICITUD	X	X	-	-
RESPUESTA	X	X	X	-
VACÍO	Solo para generar un pin de reinicio	-	X	X

Nota: Adaptado de (Shelby et al., 2014)

6.1.4 Modelo Solicitud/Respuesta

La semántica de solicitud y respuesta de CoAP se transporta en mensajes CoAP, que incluyen un Código de método o un Código de respuesta. CoAP utiliza los siguientes métodos para realizar mensajes de solicitud (Shelby et al., 2014):

- GET: Recupera una representación del recurso identificado por el URI.
- PUT: Solicita se actualice el recurso identificado por el URI.
- POST: Solicita que se procese la representación incluida en la solicitud.
- DELETE: Solicita que se elimine el recurso identificado por el URI de solicitud.

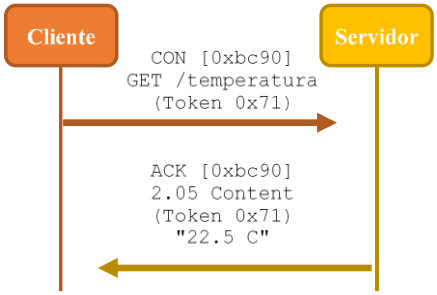
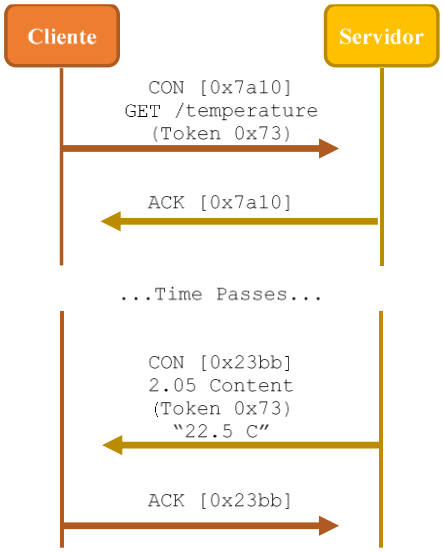
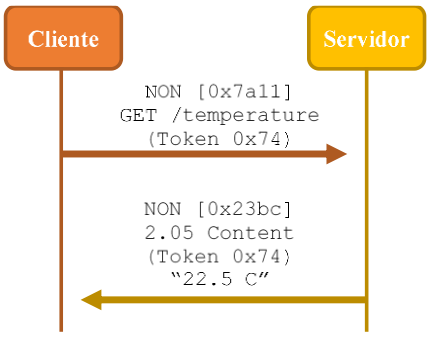
El protocolo CoAP incluye las siguientes clases de códigos en sus mensajes para dar respuesta a solicitudes (Shelby et al., 2014):

- Éxito 2.xx: Indica que la solicitud del cliente se recibió, entendió y aceptó con éxito. Ejemplo, respuesta 2.05 (Contenido)
- Error de cliente 4.xx: Indican error en la solicitud del cliente. Son aplicables a cualquier método de solicitud. Ejemplo, respuesta 4.05 (Método no permitido).
- Error de servidor 5.xx: Indica casos en los que el servidor es consciente de que ha cometido un error o es incapaz de realizar la solicitud. Ejemplo, respuesta 5.00 (Error interno del servidor)

En la tabla 7, se presentan algunos modelos de Solicitud/Respuesta.

Tabla 7.

Modelo de Solicitud/Respuesta

Modelo Solicitud/Respuesta	Ejemplo
Respuesta Superpuesta: Si el cliente envía una solicitud GET en un mensaje (CON) o (NON) a un servidor y, si está disponible la respuesta, se transporta en el mensaje de confirmación (ACK) resultante, como en la figura 16.	 *Figura 16.* Solicitud GET con respuesta superpuesta
Respuesta separada o asíncrona: Si el servidor no puede responder de inmediato a una solicitud que se incluye en un mensaje (CON), simplemente responde con un (ACK) vacío para que el cliente pueda detener la retransmisión de la solicitud. Cuando la respuesta está lista, el servidor la envía en un nuevo mensaje (CON) y el cliente responde con un mensaje (ACK), como se observa en la figura 17.	 *Figura 17.* Una solicitud GET con una respuesta separada
Respuesta en mensaje (NON): Si el cliente envía una solicitud en un mensaje (NON), la respuesta se envía utilizando un nuevo mensaje (NON), aunque el servidor puede enviar un mensaje (CON), como se ilustra en la figura 18.	 *Figura 18.* Una solicitud y una respuesta llevadas en mensajes no confirmables

Nota: Adaptación de (Shelby et al., 2014)

6.1.5 Almacenamiento en cache

CoAP provee un método de almacenamiento simple en caché, que consiste en reutilizar un mensaje de respuesta anterior para satisfacer una nueva solicitud (Shelby et al., 2014). En ciertas condiciones, un mensaje de respuesta almacenado puede reutilizarse sin la necesidad de un mensaje de solicitud. Para que una respuesta anterior sea reutilizada debe cumplir las siguientes condiciones:

- Que el método de solicitud presentado coincida con el método utilizado para obtener la respuesta almacenada.
- Que todas las opciones coinciden entre la solicitud presentada y la solicitud utilizada para obtener la respuesta almacenada, excepto las opciones Max-Age o ETag.
- Que la respuesta almacenada es nueva o aún no ha expirado.

6.1.6 Formato del mensaje de CoAP (Constrained Application Protocol)

Los mensajes CoAP están codificados en un formato binario simple, que comienza con una cabecera de 4 bytes de tamaño fijo, conformada por los campos: Ver, T, TKL, Code y Message ID (como se observa en la figura 19). En esta primera parte de la cabecera se define el tipo de mensaje que se está enviando.

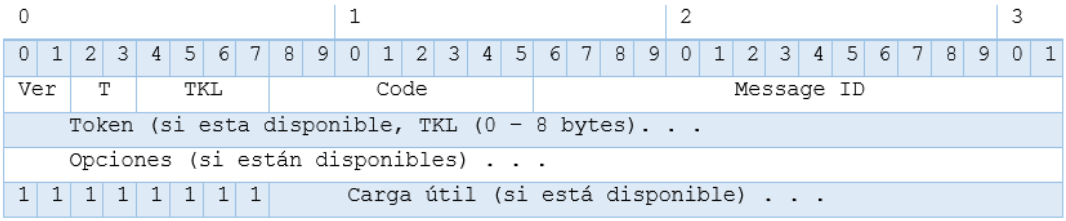


Figura 19. Formato del mensaje de CoAP

Adaptado figura 7 (Shelby et al., 2014)

Dónde:

Ver: es un entero sin signo de 2 bits que indica la versión de CoAP.

T: es un entero sin signo de 2 bits que indica el tipo de mensaje (*CON, NON, etc.*).

TKL: es la longitud del token. Tiene una longitud de 4 bits.

Code: es el código de la solicitud/respuesta de CoAP. Tiene una longitud de 8 bits.

Message ID: es la ID del mensaje. Tiene una longitud de 16 bits

Luego de la cabecera se encuentra el valor de token, cuya longitud es variable y se encuentra de 0 a 8 bytes. Seguido de una secuencia de cero o más opciones de CoAP en formato TLV (Tipo-Longitud-Valor), opcionalmente acompañado de una carga útil que ocupa el resto del datagrama.

6.1.7 Descubrimiento

En entornos M2M, donde no hay intervención humana y la configuración manual de cada nodo resulta tediosa y lenta, el descubrimiento de recursos alojados por un servidor restringido es muy importante (Shelby et al., 2014). Como se describe en (Shelby, 2012), esta funcionalidad es utilizada por un servidor CoAP para alojar directorios de recursos que contienen identificadores (URI), enlaces y atributos de recursos. Estos directorios de recursos permiten a los clientes acceder a otros recursos por medio de un mensaje de solicitud a una interfaz de búsqueda del directorio de recursos.

6.1.8 Proxing

En (Shelby et al., 2014), se define el proxy como un punto final CoAP que sirve para el reenvió de solicitudes, retransmisión de respuestas, almacenamiento en caché y traducción de protocolos (proxy cruzado), entre otros. Según la posición en el diseño del reenvío de solicitudes, se establecen dos formas de proxy: de reenvío e inverso. El proxy de reenvió se encarga de

recuperar recursos en nombre de los clientes y el proxy inverso recibe las solicitudes como si fuera un servidor y da respuesta en nombre de estos, los clientes no perciben que se trata de un proxy (Shelby et al., 2014).

6.1.9 Características extendidas

Existen extensiones que agregan características que aumentan las capacidades del protocolo CoAP. A continuación, se presentan algunas de estas:

- **RFC 7641 - Observing resources in the constrained application protocol (CoAP):** Según (Hartke, 2015), esta extensión permite a los clientes CoAP observar un recurso, con el fin de obtener una representación y mantenerla actualizada por el servidor durante un período de tiempo. Esto se logra al agregar la opción Observe al método GET y su comportamiento varía dependiendo de si vienen agregado en una solicitud o una respuesta. Con el uso de esta extensión se reduce el número de solicitudes y respuestas entre el cliente y servidor, permitiendo así la descongestión de la red.
- **RFC 7959 - Block-Wise transfers in the constrained application protocol (CoAP):** Esta extensión define las opciones Block1 y Block2 de CoAP que permiten enviar grandes cargas útiles cuando una representación de recursos es más grande de lo que se puede transferir en la carga útil de un solo datagrama CoAP. El objetivo principal es evitar la necesidad de crear un estado de conversación en el servidor para solicitudes GET en bloque (Carsten Bormann & Shelby, 2016).
- **RFC 7967 - Constrained application protocol (CoAP) option for no server response:** Esta extensión introduce la opción “No response”, que permite al cliente manifestar al

servidor su desinterés por determinada respuesta a una solicitud en particular. La opción "No response" puede ser utilizada con los métodos PUT y POST (Vardaro et al., 2016).

- **RFC 8132 - PATCH and FETCH Methods for the constrained application protocol (CoAP):** Esta especificación define los métodos FETCH y PATCH, que se utilizan para acceder y actualizar partes de un recurso respectivamente (van der Stok, Bormann, & Sehgal, 2017).
- **RFC 8323 - CoAP (Constrained application protocol) over TCP, TLS, and WebSockets):** Esta extensión introduce el uso de CoAP sobre TCP, enlaces TLS para la seguridad del protocolo y transporte sobre WebSockets. Esta extensión es aplicable para los casos en donde los paquetes UDP no son enviados por la infraestructura de red (C. Bormann et al., 2018).
- **RFC 8613 - Object security for constrained RESTful environments (OSCORE):** Esta extensión de CoAP admite las funcionalidades de las opciones de Observe (Hartke, 2015), Block-Wise (Carsten Bormann & Shelby, 2016), No Reponse (Vardaro et al., 2016), los métodos PATCH y FETCH (van der Stok et al., 2017), así como también, altera el procesamiento de CoAP al actualizar el número de opciones de CoAP y el registro IANA. Se encarga de proteger los mensajes de solicitud/respuesta de la capa de aplicación de extremo a extremo a través de nodos intermedios, como los servidores proxy de CoAP y los traductores de protocolo cruzado, incluidos los servidores proxy HTTP a CoAP entre los puntos finales (Selander, Mattsson, Palombini, & Seitz, 2019).

6.1.10 Implementaciones

CoAP debido a su simplicidad se puede implementar desde cero en una aplicación simple, o se pueden utilizar implementaciones ya existentes disponibles para varias plataformas, en aplicaciones que buscan satisfacer requisitos específicos. Muchas de las implementaciones son de código abierto y están dirigidas a varios entornos, en la tabla 8 se presentan algunas de estas librerías junto con características como: extensiones, plataforma de destino, lenguaje de programación, e interoperabilidad.

Según (C Bormann, 2016) las implementaciones utilizadas en dispositivos restringidos son: Erbium, libcoap, tinydtls, SMCP, microcoap, cantcoap, Lobaro CoAP, MR-CoAP, wakaama, coap-shepherd y coap-node. Así mismo, en proyectos para redes de tipo IEEE 802.15.4 como en (Vo et al., 2018) usan la librería Erbium, en (Tajmajer, Sadkowski, & Winiecki, 2015) utilizan la biblioteca CoAPthon, en el proyecto (Hermanudin, Ekadiyanto, & Sari, 2019) implementan con la librería aiocoap y en (Sulaeman, Ekadiyanto, & Sari, 2016) usan la biblioteca libcoap; estas librerías se describen en la tabla 8.

Tabla 8.

Implementaciones CoAP

Implementaciones	Descripción
Californium	Es una biblioteca Java, desarrollado por la fundación Eclipse con licencia EPL y EDL, dirigido para servicios de back-end (lado del servidor) y para dispositivos integrados que ejecutan Linux (controladores de domótica, controladores de fábricas inteligentes y teléfonos móviles). Sigue el RFC 7252. Admite algunas extensiones como: el modo Observe, las transferencias Block-Wise y el directorio de recursos. Para la seguridad implementa DTLS 1.2 a través del submódulo Scandium. (Eclipse Foundation, 2019)
Libcoap	Es una biblioteca escrita en C, con licencia BSD/GPL diseñada para dispositivos embebidos con pocos recursos (con sistema operativo Contiki o LWIP), como también en un sistema grande con POSIX. Compatible con el RFC 7252 oficial para el cliente y el servidor, además es compatible con varias extensiones como: el modo Observe, las transferencias de Block-Wise y el directorio de recursos, opción para ninguna respuesta del servidor, CoAP sobre TCP, TLS y WebSockets, métodos PATCH y FETCH. Para la seguridad utiliza TLS con GnuTLS, OpenSSL o tinydtls. Además, el código fuente viene con ejemplos completos. (Iglesias-Urkia, Orive, & Urbieta, 2017)
CoAPthon	Es una biblioteca escrita en Python, con licencia MIT. Compatible con RFC7252 para el cliente y servidor, también para el proxy inverso y de reenvío. Además de las características principales, es compatible con el modo Observe, el formato Core-Link, las extensiones de las transferencias de Block-Wise y multidifusión. (Iglesias-Urkia et al., 2017)
CoAPSharp	Es una biblioteca liviana, escrita en lenguaje C# y .NET. Maneja licencia LGPL, funciona con Microsoft .NET y Micro Framework. Admite la RFC 7252 para cliente y servidor, compatible con el modo Observe, directorio de recursos, las transferencias de Block-Wise y por supuesto la seguridad. CoAPSharp fue desarrollado originalmente por EXILANT y adquirido por Idaax. (“CoAPSharp – Idaax”, 2019)
Erbium	Es una biblioteca escrita en C para el sistema operativo Contiki. Maneja licencia 3-clause BSD. Diseñado para permitirle a los pequeños dispositivos con sistema de baja potencia, conectarse a Internet. Sigue la RFC 7252 para cliente y servidor, admite el modo Observe, y las transferencias de Block-Wise. (Iglesias-Urkia et al., 2017)

jCoAP	Es una biblioteca de Java, con licencia Apache License 2.0. Está diseñado para funcionar en dispositivos pequeños y sistemas integrados como teléfonos inteligentes basados en Java (Android). Compatible con la RFC 7252 para el cliente y el servidor. Admite el modo Observe y las transferencias de Block-Wise. La Comunicación grupal, aún no se ha implementado, ya que esto todavía está en progreso. (Konieczek, Rethfeldt, Golatowski, & Timmermann, 2015)
aiocoap	Es una biblioteca escrita en Phyton 3, con licencia MIT. Sigue la RFC 7252 para cliente y servidor de forma parcial, compatible de forma parcial con las extensiones: el modo Observe, las transferencias de Block-Wise, CoAP sobre TCP y TLS, métodos PATCH y FETCH, directorio de recursos y seguridad OSCORE. Compatible con la opción sin respuesta del servidor. (Amsüss, 2014)
SMCP	Es una biblioteca escrita en C. Diseñada para dispositivos integrados, desde sensores hasta dispositivos basados en Linux. Admite la RFC 7252, compatible con el cliente y el servidor. Admite sockets BSD y µIP. Admite el modo Observe y los grupos de multidifusión. Cuenta con soporte experimental para DTLS. Incluye smcpctl que es una herramienta de línea de comando para navegar y configurar nodos CoAP. (Iglesias-Urkia et al., 2017)
microcoap	Es una biblioteca de C, dirigida a pequeños microcontroladores. El código fuente proporciona ejemplos para POSIX y Arduino. Sigue la RFC 7252 pero solo admite el lado del servidor con características limitadas. No admite solicitudes DELETE, solo GET, PUT y POST, no admite reintentos y los ACK únicamente superpuestos. (Iglesias-Urkia et al., 2017)

Nota: Adaptación de (“Constrained Application Protocol”, 2019)

6.1.11 Casos de aplicación

El protocolo CoAP aplicado en WSN se puede adaptar a varias áreas, debido a que es un protocolo ligero, optimiza el consumo de energía y permite su utilización en dispositivos restringidos, habilitándolos para conectarse a la red IoT. A continuación, se muestran algunos casos de uso en diferentes ámbitos de aplicación en la tabla 9.

Tabla 9.

Casos de aplicación CoAP

Áreas	Ámbito de aplicación	Casos de uso
HOGARES INTELIGENTES	CoAP es usado en el hogar para automatizar actividades como: • El uso eficiente de la energía y medición de consumo de los electrodomésticos. (Teklemariam, Hoebeke, Van Den Abeele, Moerman, & Demeester, 2014) (Belcredi, Modernell, Sosa, Steinfeld, & Silveira, 2015) • La medición de la temperatura ambiente. (Ravi Kishore Kodali, Krishna Yogi, Sharan Sai, & Honey Domma, 2018) • Gestionar la apertura y cerrado de persianas de acuerdo con la temperatura, luz o presencia de personas en determinado espacio. (Castro Heredia, 2014)	• Administración de la energía en un hogar automatizado, por medio de redes de Sensores/ Actuadores que ejercen la función de las bombillas e interruptores, permitiendo que desde un objeto conectado a Internet como un teléfono móvil se ejecuten acciones como encendido y apagado de luces o consulta de registro de consumo (Teklemariam et al., 2014)
EDIFICIOS INTELIGENTES	CoAP es usado en edificios para automatizar actividades como: • Monitoreo de puertas, sistemas acondicionado y ascensores. (Van den Abeele, Moerman, Demeester, & Hoebeke, 2017) • Gestión del uso y consumo de electricidad. (Vo et al., 2018)	• Sistema inteligente monitoreado a través del Internet con servicios como: gestión de puertas, iluminación, control de clima, ascensores y la supervisión de presencia en ciertas áreas. (Van den Abeele et al., 2017) • Prototipo de sistema de sensores y actuadores para un edificio de varias plantas, con el fin de monitorear y controlar en tiempo real el uso eléctrico del edificio desde cualquier parte, con dispositivos conectados a Internet a través de comunicación por un proxy HTTP-COAP. (Vo et al., 2018)
CIUDADES INTELIGENTES	CoAP es usado en ciudades para actividades como: • Estacionamiento Inteligente de vehículos. (Kayal & Perros, 2017) • Medición del tráfico de transporte. (Rajasekaran, Janardhan, & Chander, 2013)	• Sistema de cobro de peaje inteligente implementado en la India, en el cual las cabinas y los vehículos están equipados con sensores, monitorean el flujo de entrada y salida de los vehículos y estos sensores notifican periódicamente a los

		usuarios registrados con actualizaciones de tráfico. (Rajasekaran et al., 2013)
INDUSTRIA	CoAP es usado en la Industria para automatizar actividades como: • Monitoreo de temperatura en entornos industriales. (Sauch Polo, 2017) Monitoreo y control de oleoductos. (Bahia & Campista, 2017) • Control de pedidos y uso eficiente de la energía eléctrica utilizada en el proceso. (Bahia & Campista, 2017)	• Sistema de monitoreo y control de oleoductos receptores de petróleo en terminales de almacenamiento, con la función de monitorear el flujo en el oleoducto que alimenta tanques, bombas de transferencia y otras tuberías. (Bahia & Campista, 2017) • Mecanismo de control de la demanda de pedidos, que garantiza la disponibilidad para clientes prioritarios en un escenario industrial de IoT. Provee la reducción de energía y del ciclo de trabajo de transmisión. (Bahia & Campista, 2017)
AGRICULTURA	CoAP es usado en la Agricultura para automatizar actividades como: • Monitoreo del clima en cultivos. (Cardozo, Camps, & Martín, 2015)	• Implementación de una red de sensores inalámbricos, para monitoreo de condiciones micro climáticas en cultivos de cítricos. Los aspectos que se supervisan incluyen humedad y temperatura del aire con el propósito de reducir el impacto de las heladas emitiendo alertas como la humedad del suelo para la racionalización del riego y permitiendo acciones postcosecha. (Cardozo et al., 2015)
SALUD	CoAP es usado en el sector Salud en actividades como: • Uso de dispositivos de salud personal (Oryema, Kim, Li, & Park, 2017) • Localización y seguimiento a pacientes. (Pierleoni et al., 2016)	• Sistema de red de sensores con conectividad de radiofrecuencia, para localización o seguimiento de pacientes y equipos médicos, que permiten detectar la habitación o el lugar donde se encuentran los objetivos. (Pierleoni et al., 2016)

6.1.12. Otros aspectos

6.1.12.1 Seguridad La seguridad para el protocolo CoAP se aplica en la capa de transporte sobre el protocolo UDP y el cifrado se realiza utilizando DTLS (Datagram Transport Layer Security) y algunas veces IPsec (Internet Protocol Security). DTLS utiliza AES (Advanced Encryption Standar) con la autenticación de código CCM (Cipher Block Chaining-Message) para proveer confidencialidad, integridad, autenticación y no repudio (R. A. Rahman & Shah, 2016).

Una reciente actualización modificó aspectos de seguridad en la RFC 7252, considerando una extensión del protocolo CoAP, introduciendo OSCORE (Object Security for Constrained RESTful Environments) [RFC8613]. Protocolo encargado de proteger las interacciones RESTful como el método de solicitud, el mensaje de solicitud, la carga útil del mensaje, etc. OSCORE puede usarse junto con los protocolos TLS o DTLS sobre uno o más saltos durante el envío de extremo a extremo, siendo transportados los mensajes ya sea con HTTPS o CoAP simple (Selander et al., 2019).

Sigue siendo un desafío la importancia de mantener seguros los datos que se transportan a través de Internet en redes compuestas por dispositivos IoT con recursos limitados; es por ello que varios estudios y artículos presentan diferentes enfoques orientados a la seguridad de estas redes en conjunto con el protocolo CoAP, como los presentados a continuación:

En (Jorge David de Hoz, Saldana, Fernandez-Navajas, & Ruiz-Mas, 2019) se estudia la viabilidad de incorporar el esquema de desacoplamiento de seguridad IoTsafe para entornos restringidos junto con el Protocolo de Aplicación Restringida CoAP. Con el esquema se propone dividir los componentes de seguridad y autenticación, dejando a cargo de estos a una capa inferior de comunicación IoTsafe, la cual podría ser manejada y actualizada por entidades

externas encargadas de su correcto funcionamiento. En el artículo se concluyó factible el desacoplamiento de la seguridad, además se documentaron los beneficios que este enfoque puede proporcionar a dispositivos IoT de tipo IEEE 802.15.4.

En (Nathi & Sutar, 2019) los autores afirman que el uso de TLS, DTLS y el mecanismo handshaking; ocasionan recursos costosos de computación debido a la complejidad de los algoritmos empleados. En el artículo se plantea manejar un algoritmo ligero e incorporar dentro del hardware durante la fabricación de los dispositivos IoT, específicamente en la memoria ROM, 2 claves privadas: PSK1 usada como un componente de autenticación y PSK2 para cifrado seguro y descifrado de los datos. Se obtuvo como resultado menos consumo de recursos y reducción del tráfico.

En (Roselin, Nanda, Nepal, He, & Wright, 2019), se propone un enfoque de seguridad basado en el modelo ML (Machine learning) para evitar ataques durante el acceso remoto a un servidor CoAP y las posibles vulnerabilidades de suplantación de identidad en las solicitudes CoAP, siendo esta una limitación de seguridad para el protocolo DTLS y el firewall, además se presentan en la mayoría de las implementaciones de CoAP.

En (Fournaris, Giannoulis, & Koulamas, 2019), se propone realizar la evaluación de la seguridad de extremo a extremo de CoAP para redes de sensores inalámbricos restringidos, integrando DTLS con el sistema operativo Contiki OS (versión 3.x). El mecanismo propuesto pretende evaluar la velocidad y consumo de energía, así mismo extraer aspectos de seguridad importantes; demostrando que un nivel alto de seguridad se logra a un costo altamente considerable de energía al establecer un canal seguro CoAP debido al mecanismo handshaking de DTLS y se estima optimo solo cuando los canales seguros de CoAP no son establecidos con frecuencia.

6.1.12.2 Enrutamiento CoAP utiliza como protocolo de enrutamiento RPL, diseñado para redes restringidas. Además, RPL posee un modo de operación que puede ser utilizado por nodos CoAP, el cual se define en la RFC 7390 como modo Storing (almacenamiento) con soporte de multidifusión. Este modo de operación se caracteriza porque los nodos hijos envían mensajes DAO (mensajes de Objeto de anuncio de destino) por unicast, al nodo padre o padre seleccionado, propagando información de direccionamiento aguas arriba, alimentando las tablas de ruteo de los nodos padre (de Mula, Ferrari, & Firme, 2011).

Entonces, los mensajes RPL DAO pueden ser utilizados por los nodos CoAP (enturadores o nodos hijos RPL) para anunciar a los enrutadores principales su pertenencia a grupos de multidifusión IP; también se utilizan para enrutar las solicitudes CoAP de multidifusión IP a través de múltiples saltos a los servidores CoAP correctos (A. Rahman & Dijk, 2014)

De igual forma, se puede utilizar el mecanismo DAO para transmitir información sobre la pertenencia a un grupo de multidifusión IP a un enrutador de borde, si el enrutador de borde es también el nodo raíz RPL DODAG. Esta particularidad es útil, porque el enrutador perimetral aprende qué tráfico de multidifusión IP debe pasar de la red troncal a la subred LLN, ayudando a evitar la congestión de la subred (A. Rahman & Dijk, 2014).

6.2 Protocolo MQTT (Message Queue Telemetry Transport)

Es un protocolo diseñado para el transporte de mensajes en entornos restringidos, maneja un patrón de mensajes Publicación/Suscripción; su finalidad es reducir el ancho de banda y los recursos de los dispositivos que lo utilizan, además ofrece confiabilidad y seguridad en la

entrega de los mensajes (Sciarrone & Sauro, 2017). Es apropiado en comunicaciones máquina a máquina (M2M) e internet de las cosas IoT (ISO/IEC 20922:2016, 2016). Su uso en aplicaciones móviles es ideal por el poco consumo de energía, el tamaño reducido de los paquetes de datos y la asignación eficiente de los mensajes a los destinatarios (Shiva Shankar, S, S, & Sowmya, 2019).

Sus principales características según (Naik, 2017) son:

- Preparado por OASIS y adoptado por los organismos nacionales ISO e IEC
- Puerto predeterminado 1883/8883 (TLS/SSL).
- Licencia Open Source.
- Con Arquitectura Cliente/Bróker o Cliente/Servidor.
- Maneja un tamaño de cabecera de 2 byte para especificar el tipo de mensaje.
- Emplea una carga útil binaria. La aplicación puede tener cualquier formato, mientras los clientes de destino puedan analizar la carga útil.
- El tamaño de los mensajes es reducido, con un máximo de 256 MB.
- El protocolo utilizado para transporte es TCP, maneja también el protocolo UDP con la especificación MQTT-SN.
- La seguridad es soportada por los protocolos TLS/SSL.
- Proporciona 3 niveles de calidad de servicio QoS 0, QoS 1y QoS 2.

6.2.1 Arquitectura y funcionamiento

La arquitectura del protocolo MQTT está orientada a la topología estrella como se aprecia en la figura 20. Según (Arco Castillo & Navarro Ortiz, 2019) y tiene los siguientes componentes:

- Bróker: Actúa como servidor o nodo central entre los clientes, es el encargado de transmitir mensajes entre ellos, mantener activo el canal de comunicación y la administración de la red.
- Clientes: Entidades que actúan como publicadores y/o suscriptores sobre un tema.
- Mensaje: Unidad de datos o información que se transmite o recibe sobre un tema.
- Tópico: Tema del cual los clientes pueden publicar información o recibirla si se encuentra suscrito a éste.

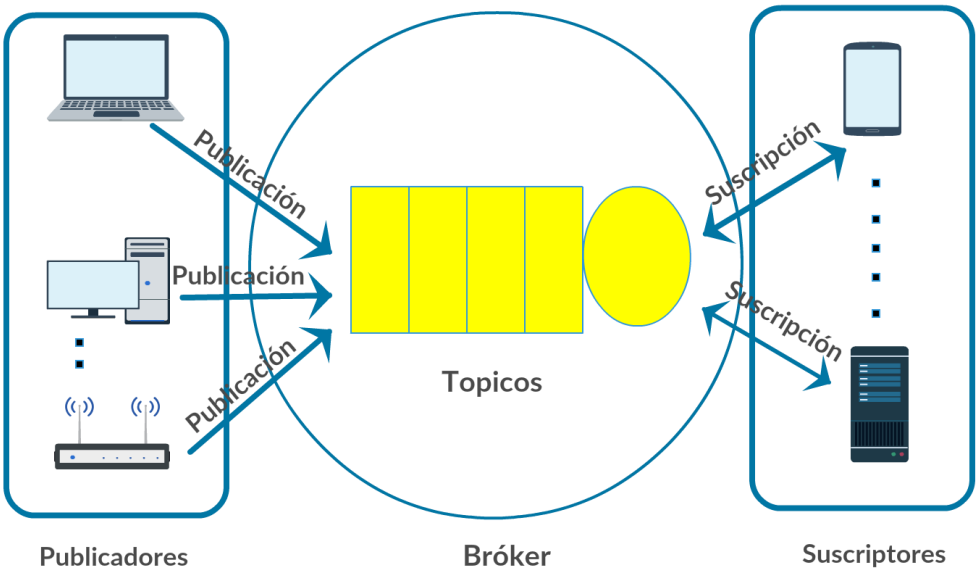


Figura 20. Arquitectura del protocolo MQTT
Adaptado de la figura 1 de (Shiva Shankar et al., 2019)

Para establecer una comunicación entre el Bróker y los Clientes, se inicia una sesión que permanece activa siempre que el Bróker reciba y gestione información de hasta aproximadamente 1000 clientes (Sciarrone & Sauro, 2017). Los clientes que requieran recibir información deben suscribirse a un tópico determinado, el cual contiene datos que otro cliente

habrá publicado y enviado al Bróker, encargado de redireccionar los paquetes de datos a los destinatarios suscritos. (Arco Castillo & Navarro Ortiz, 2019).

En el modelo Publicación/Suscripción del protocolo MQTT, los clientes no se conocen entre ellos ni tienen conocimiento de quien publica o se suscribe a determinado tópico, lo que produce un desacoplamiento en tiempo al no tener que ejecutarse a la vez y un desacoplamiento de sincronismo porque no deben detener las operaciones que esté realizando. Por este motivo es uno de los modelos que más se adaptan a las necesidades de los sistemas de IoT. (Moreno Cerdà, 2018)

6.2.2 Formato de los paquetes de control MQTT (Message Queue Telemetry Transport)

La función del protocolo MQTT es transmitir mensajes en un orden determinado, estos mensajes se intercambian en un formato llamado paquete de control. Su estructura está conformada por 3 partes como se visualiza en la figura 21. Las cuales se describen en esta sección.

Cabecera Fija - Presente en todos los paquetes de control MQTT.
Cabecera Variable - Presente en alguno de los paquetes de control.
Carga Útil - Presente en alguno de los paquetes de control.

Figura 21. Formato de los paquetes de control MQTT
Adaptación a partir de la figura 2-1 de (Montilla Pérez, 2018)

6.2.2.1 Cabecera fija Todos los paquetes de control MQTT tienen una cabecera fija, la cual esta subdividida por tres componentes distribuidos en dos octetos: el primer octeto del bite 7 al 4 contiene el tipo de paquete de control, del bit 3 al 0 contiene las banderas específicas para

cada paquete de control y el segundo octeto comprende la longitud restante (Montilla Pérez, 2018). Como se aprecia en la figura 22.

Bite	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Tipo de paquete de control MQTT				Banderas específicas para cada paquete de control MQTT			
Byte 2…	Longitud Restante							

Figura 22. Formato de la cabecera fija

Adaptación a partir de la figura 2.2 de (ISO/IEC 20922:2016, 2016)

- **Tipos de paquetes de control MQTT:** Equivalen a un fragmento de la cabecera fija en el cual se especifica el tipo de paquete del que trata, ocupa cuatro bits [7-4] del byte 1. Existen 14 tipos de paquetes de control: Connect, Connack, Publish, Puback, Pubrec, Pubrel, Pubcomp, Subscribe, Suback, Unsubscribe, Unsuback, Pingreq, Pingresp, Disconnect. En la tabla 10, se presenta una breve descripción de los 6 tipos más relevantes según (Yuan, 2017); para obtener más información acerca de cada uno de ellos, visite la sección 3 del Estándar ISO/IEC 20922.

Tabla 10.

Tipos de paquetes de control MQTT

Paquete	Descripción	Parámetros
CONNECT	Cliente pide conexión con el Servidor	**CleanSession:** Indica persistencia de la conexión. **Nombre de usuario y contraseña:** Credenciales de autenticación y autorización del Bróker. **LastWillTopic:** Mensaje enviado cuando la termina inesperadamente la sesión. **LastWillQos:** La QoS del mensaje de "último deseo". (Consultar sección 7.2.4) **LastWillMessage:** El mensaje en sí de "último deseo". **KeepAlive:** Lapso en que el cliente realiza un ping en el bróker para mantener activa la conexión.
CONNACK	Confirmación para la conexión	**SessionPresent:** Indica si la conexión ya contiene una sesión persistente. **ReturnCode:** 0 indica éxito, otros valores identifican la causa de la falla.
SUBSCRIBE	Petición para suscribirse del cliente	**Qos:** Indica con qué consistencia los mensajes, en este tema, necesitan ser entregados a los clientes. (Consultar sección 7.2.4) **Tema:** Tema es un formato determinado**.** (Consultar la sección 7.2.3).
SUBACK	Confirmación de la suscripción	**ReturnCode:** Códigos de retorno para cada uno de los temas en el comando SUBSCRIBE. Los valores de retorno son los siguientes: Valores 0 a 2: éxito como nivel de QoS correspondiente. Valor 128: falla.
UNSUBSCRIBE	Petición para darse de baja	**Tema:** Este parámetro puede repetirse en varios temas.
PUBLISH	Mensaje de publicación	**TopicName:** El tema en el cual se publica el mensaje. **Qos:** El nivel de calidad de servicio de la entrega del mensaje. **RetainFlag:** Indica si el bróker retendrá el mensaje como el último mensaje conocido de este tema.

Nota: Adaptación a partir de (Yuan, 2017)

- **Banderas para cada paquete de control:** Una bandera (flag) ocupa los cuatro bits del [3-0] del byte 1. Existe una bandera específica para cada tipo de paquete de control. Casi todos los paquetes de control se encuentran en un estado reservado, lo que indica que se pueden usar en un futuro. Excepto el paquete PUBLISH, el cual maneja los siguientes valores: **DUP:** Entrega duplicada de un paquete de control PUBLISH, **QoS:** Calidad del servicio definida para PUBLISH y **RETAIN:** Bandera de retención. (ISO/IEC 20922:2016, 2016)
- **Longitud restante:** La longitud restante ocupa el segundo octeto en el bite 2 de la cabecera fija, indica el número de octetos restantes para el paquete actual, incluyendo los datos de la cabecera variable y la carga útil. (Montilla Pérez, 2018)

6.2.2.2 Cabecera Variable El encabezado variable lo contiene algunos tipos de paquetes de control, su contenido difiere de acuerdo con el tipo de paquete. Los paquetes poseen un campo llamado identificador de paquete que ocupa 2 byte. El identificador no se repite entre paquetes. Sin embargo, éste puede volver a usarse después de ser liberado por un paquete cuando se confirme su entrega. (Montilla Pérez, 2018)

6.2.2.3 Carga útil La carga útil o PAYLOAD comprende la información que se requiere transmitir en el mensaje y es de longitud variable, se encuentra solo en los siguientes paquetes de control MQTT: CONNECT, SUBSCRIBE, SUBACK y UNSUBSCRIBE, para el paquete PUBLISH es opcional. (Arco Castillo & Navarro Ortiz, 2019)

6.2.3 Caracteres utilizados en los tópicos

Los caracteres, su uso y la forma en que se presentan los tópicos se aprecian en la tabla 11:

Tabla 11.

Caracteres de los tópicos en MQTT

Carácter	Uso	Sintaxis
/	Separa los temas entre sí de forma jerárquica.	"Torre/Piso1/Sensor1"
#	Determina un nivel, permitiendo la suscripción a más de un tema a la vez.	Al suscribirse a "Torre/Piso1/Sección/#", se reciben mensajes publicados por: "Torre/Piso1/Sección1/Sensor1" "Torre/Piso1/Sección1/Sensor2"
+	Permite la suscripción en un mismo nivel	Al suscribirse a "Torre/+", se reciben los mensajes de "/Torre/Piso" y "Casa/Piso2", pero no de "Casa/Piso2/Sección1".

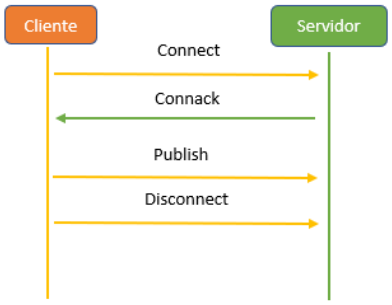
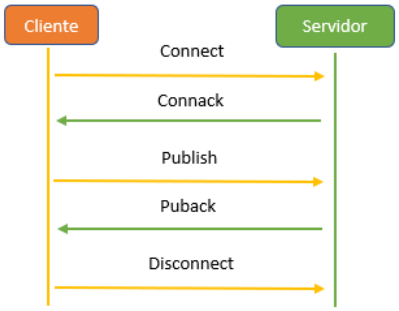
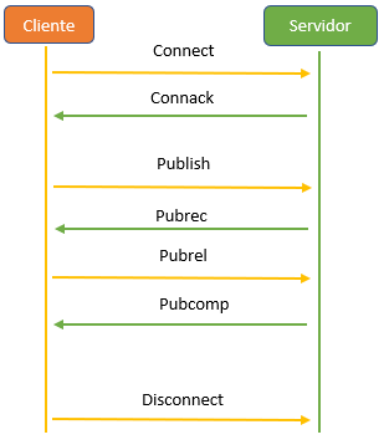
Nota: Adaptación a partir de (Montilla Pérez, 2018)

6.2.4 Calidades de servicio QoS

En esta sección se presenta una breve descripción de las tres calidades del servicio que abarca el protocolo MQTT, las cuales se aprecian en la Tabla 12. (Montilla Pérez, 2018)

Tabla 12.

Niveles de QoS en MQTT

Niveles QoS	Dirección de mensajes para QoS
Como máximo una vez (QoS 0): El nivel de calidad QoS 0 no garantiza que el paquete PUBLISH llegue correctamente al receptor, debido a que solo es enviado una vez y no se obtiene confirmación para la publicación (PUBACK) y tampoco se permite el reenvío de paquetes. Por lo que no se garantiza que el paquete haya llegado a su destino.(Montilla Pérez, 2018)	 *Figura 23.* Transmisión de mensaje con QoS 0 Adaptado de la figura 2-1 (Montilla Pérez, 2018)
Al menos una vez (QoS 1): El nivel de calidad QoS 1 garantiza que el paquete PUBLISH llegue al menos una vez al receptor, pero a su vez es propenso a que se reciba un mensaje duplicado. El transmisor puede enviar varios paquetes PUBLISH y recibir confirmación para la publicación (PUBACK).(Montilla Pérez, 2018)	 *Figura 24.* Transmisión de mensaje con QoS 1 Adaptado de la figura 2-2 (Montilla Pérez, 2018)
Exactamente una vez (QoS 2): En el nivel de calidad QoS 2 se garantiza que no exista perdida ni duplicación de paquetes PUBLISH, realizando la entrega del mensaje una única vez. El paquete PUBLIC se envía con un identificador, el cual será usado por el receptor por medio de un paquete de publicación recibida (PUBREL).(Montilla Pérez, 2018)	 *Figura 25.* Transmisión de mensaje con QoS 2 Adaptado de la figura 2-3(Montilla Pérez, 2018)

6.2.5 Características extendidas

- **MQTT sobre WebSocket** El protocolo MQTT posee una extensión que permite utilizar el protocolo WebSocket como red de transporte, en donde los paquetes de control MQTT se envían en tramas de datos binarios de WebSocket y la comunicación se establece por defecto por el puerto oficial 1883(ISO/IEC 20922:2016, 2016)

6.2.6 Implementaciones

Las aplicaciones en MQTT que requieren satisfacer determinados requisitos, pueden optar por hacer uso las diferentes implementaciones existentes tanto para Clientes como para Brókers, las cuales se pueden encontrar libres y comerciales. En la tabla 13 se mencionan algunas de las librerías más utilizadas y sus características.

Tabla 13.

Implementaciones MQTT

Implementación	Descripción
Mosquitto	Es un Bróker de mensajería escrito en lenguaje C, desarrollado por la Fundación Eclipse y patrocinado por Cedalo, con Licencia publica de Eclipse 1.0. Es compatible con MQTT v3.1, v3.1.1 y v5.0. Ampliamente utilizado por su ligereza, en varios ambientes incluidos los de bajos recursos. Proporciona también una librería en C para implementar clientes MQTT y para la seguridad admite SSL/TLS basado en certificados y en claves compartidas previamente. (Eclipse Foundation, 2018)
MQTTRoute	Es un Bróker de alto rendimiento escrito en lenguaje C y Python, es una versión premium gratuita desarrollada por Webywise Network. Compatible con MQTT v2.1, v3.1 y v3.1.1, funciona con todos los clientes MQTT estándar, admite lo niveles MQTT QoS 0, 1 y 2, soporta conexiones persistentes y limpias, se puede implementar de forma segura con servidores en la nube privados, permite integración flexible con bases de datos y proporciona transferencia de datos segura con autenticación y cifrado por medio del certificado TLS/SSL. (Bevywise Networks Inc, 2020)

ActiveMQ	Es un servidor de mensajería escrito en lenguaje Java, admite varios protocolos y clientes con diversos lenguajes como C, C++, Python, entre otros. Desarrollado por la Fundación Apache bajo la licencia Apache 2.0, es compatible con MQTT v3.1.1, maneja un modelo de direccionamiento JMS, provee clusterización de alto rendimiento para ofrecer disponibilidad y distribución de carga, Cliente-Servidor, con una comunicación basada en pares. Proporciona seguridad con JASS y configuraciones sencillas en XML para autenticación y autorización. (Apache ActiveMQ, 2019)
HiveMQ	Es un Bróker MQTT de mensajería escrito en lenguaje Java, desarrollado por dc-square GmbH, con licencia comercial, de igual forma maneja otro tipo de Bróker con licencia Apache 2.0 de código abierto, es compatible con MQTT v3.1, v3.1.1 y v5.0. Diseñado para implementaciones en la nube tanto privadas, hibridas y públicas; utiliza la tecnología PUSH para enviar y recibir datos de manera rápida desde los dispositivos conectados, a nivel empresarial escala hasta 10 millones de dispositivos. Para seguridad utiliza certificado TLS/SSL. (HiveMQ, 2020)
Paho MQTT	Es una implementación de código abierto, escrito en varios lenguajes: C, C++, Java, JavaScript, Python y Go. Desarrollado por la Fundación Eclipse, con Licencia publica de Eclipse 1.0 y licencia de distribución 1.0. Maneja el servicio Android Eclipse Paho, el cual comprende una biblioteca Cliente MQTT escrita en Java, para desarrollar aplicaciones en Android, incluye niveles efectivos de desacoplamiento entre dispositivos y aplicaciones. Para seguridad utiliza certificado TLS/SSL. (Eclipse Paho, s/f)
MQTT-C	Es una implementación de Cliente escrita en el lenguaje C, desarrollado por Liam Bindle y Demilade Adeoye, con licencia MIT, compatible con MQTT v3.1.1. Diseñado para proporcionar un cliente MQTT portátil tanto para sistemas embebidos como para PC, es ideal para sistemas integrados y microcontroladores. MQTT-C es liviano; los archivos que lo componen contienen menos de 2000 líneas y Para seguridad utiliza certificado TLS/SSL. (Bindle & Adeoye, s/f)
WolfMQTT	Es una implementación multiplataforma de Cliente escrita en lenguaje C, Desarrollado por wolfSSL, tiene doble licencia bajo GPLv2 y licencia comercial, es compatible con MQTT v3.1.1 y v5.0 y admite todos los niveles de seguridad QoS. Su diseño proporciona facilidad de portabilidad, posibilitando la compilación en nuevas plataformas. Para seguridad admite SSL/TLS utilizando la biblioteca wolfSSL. (wolfSSL Inc, 2020)

6.2.7 Casos de aplicación

El protocolo MQTT es utilizado en varias áreas de aplicación debido a su capacidad de reducir el ancho de banda y los recursos en la red. En la tabla 14 se mencionan algunos casos de uso que buscan estos requisitos en diferentes entornos.

Tabla 14.

Casos de aplicación MQTT

Áreas	Ámbito de aplicación
HOGARES INTELIGENTES	MQTT es usado en hogares inteligentes para automatizar actividades como: • Control de apertura y cerrado de puertas, encendido y apagado de bombillos, control de temperatura y humedad. (Valiente Cristancho, 2017). • Control de seguridad en el hogar con sistema contra intrusos. (Arequipa Cunalata, 2019)
EDIFICIOS INTELIGENTES	MQTT es usado en edificios inteligentes para actividades como: • Redes de alarma contra incendios. (Ravi K. Kodali & Valdas, 2019) • Sistemas de alerta contra intrusos. (Bharath, Reddy, & Dey, 2018)
CIUDADES INTELIGENTES	MQTT es usado en ciudades para actividades como: • Generación de un banco de pruebas de monitoreo de calidad del aire, incluyendo temperatura, humedad y presión. (Tanyingyong, Olsson, Hidell, Sjödin, & Ahlgren, 2018). También Monitoreo de la calidad del aire en áreas rurales y urbanas. (Candia & Varela, 2017) • Notificación temprana al sector investigativo y policial, sobre personas desaparecidas en determinadas zonas de una ciudad. (Ortiz Mejía, 2018) MQTT es utilizado en redes de tipo 802.15.4 en ciudades inteligentes para: • Medir la temperatura, humedad y luminosidad a partir de sensores ubicados en los postas de la ciudad. (Lima, 2018)
INDUSTRIA	MQTT es usado en la industria para automatizar actividades como: • Monitoreo y control de brazo robótico industrial. (Atmoko & Yang,

	2019) • Predicción de mantenimiento para máquinas industriales. (Figueiredo et al., 2018)
	MQTT es utilizado en redes de tipo 802.15.4 en la industria, en sistemas como: • IoT fPerception que mide la concentración de contaminación del aire, en operaciones portuarias y plantas de energía y calor. Mide aspectos como: humedad, presión, amoniaco, sulfato de hidrogeno, entre otros. (Libelium, 2018b)
AMBIENTE INTELIGENTE	MQTT es usado en el medio ambiente para automatizar actividades como: • Monitoreo ambiental en el mar. (Meera & Rao, 2018) • Monitoreo del consumo del agua para su uso racional. (Schreiber, Dias, Borba, Noll, & Gallo, 2019)
ENERGÍA	MQTT es usado en energía para actividades como: • Monitoreo de consumo de corriente continua en tiempo real. (Astorqui, 2016) • Monitoreo de generadores de energía para gestionar los tipos de mantenimiento y el consumo de energía. (Schreiber et al., 2019)
AGRICULTURA	MQTT es usado en la agricultura para actividades como: • Medición de humedad del suelo en plantas. (Ravi Kishore Kodali & Sahu, 2016) • Control de bombas de agua para el riego de suelos. (Ravi Kishore Kodali & Sarjerao, 2017) • Monitoreo de temperatura, humedad y luz de un invernadero. (Syafarinda et al., 2018)
SALUD	MQTT es usado en Salud para actividades como: • Monitoreo a distancia de señales biomédicas, con respecto a la actividad eléctrica del corazón por medio de Electrocardiogramas (ECG). (Sciarrone & Sauro, 2017) • Diagnóstico médico de enfermedades neurodegenerativas a partir de la EMT (estimulación magnética transcraneal). (Depari et al., 2019)

6.2.8 Otros aspectos

6.2.8.1 Seguridad Según (ISO/IEC 20922:2016, 2016), el protocolo MQTT no tiene establecido un esquema de seguridad integrado. Sin embargo, recomienda a los implementadores que requieran utilizar el protocolo, emplear TLS en el servidor por el puerto TCP 8883 para otorgar integridad y privacidad en la comunicación, junto con el estándar de cifrado avanzado AES y el estándar de cifrado de datos DES para la encriptación de mensajes. (Arco Castillo & Navarro Ortiz, 2019). O en su defecto, generar sus propios sistemas de seguridad e integrarlos con el funcionamiento del protocolo.

En consenso con lo anterior, diferentes autores han optado por crear métodos, mecanismos y esquemas de seguridad que abordan características robustas comparadas con el uso convencional de TLS y los algoritmos AES Y DES. De los cuales se mencionan algunos a continuación:

En (Thantharate, Beard, & Kankariya, 2019) los autores proponen y evalúan tres métodos para determinar la confiabilidad y eficacia del envío de parches de seguridad y actualizaciones de software por aire OTA (Over-The-Air), entre dispositivos restringidos. Uno de ellos utiliza los protocolos MQTT y COAP al mismo tiempo; en donde se emplea MQTT como protocolo base y éste a su vez utiliza encapsulado al protocolo CoAP, lo que significa que un dispositivo recibe los mensajes MQTT, pero lee los datos de una URL CoAP. Se concluye que MQTT es más confiable y rápido, y el protocolo CoAP es adecuado para transmitir grandes cantidades de datos con un buen uso de la red.

En (Kumar & Dezfouli, 2019) se presenta una implementación del protocolo QUIC (Quick UDP Internet Connections), con el protocolo MQTT, en el que se describe su arquitectura y el desarrollo de los Brókers necesarios para establecer la comunicación entre ellos; se determinó que esta combinación reduce la sobrecarga de conexión, disminuye la

latencia de entrega, reduce la utilización de procesador y memoria, en comparación con el uso convencional de MQTT con TCP.

En (Almazroi, 2019) se propone un mecanismo de seguridad con la combinación de los protocolos MQTT y HTTP, que proporciona seguridad de extremo a extremo, reduciendo la posibilidad de que ocurran ataque de denegación de servicios DoS en los servidores MQTT. Con la intervención del protocolo Zigbee que actúa como intermediario para realizar la conversión de los mensajes entre los protocolos MQTT y HTTP.

En (Nolan, 2018), se propone un esquema de seguridad en donde se implementa un KMS (Key Management Service), este servicio de gestión de claves actúa como un servidor central encargado de gestionar el cifrado de la carga útil y autenticación para el acceso y comunicación entre Clientes y Bróker. Asignando a los clientes un ID, una contraseña y un tópico privado para la comunicación entre el cliente y el KMS, además provee claves con fecha de vencimiento, para cifrar y descifrar mensajes recibidos o enviados entre los clientes y el Bróker.

6.3 Protocolo AMQP (Advanced Message Queuing Protocol)

AMQP es un protocolo de internet abierto, desarrollado para el intercambio de mensajes empresariales entre diferentes plataformas y estandarizado por OASIS en el documento (ISO/IEC 19464:2014, 2014). Es un estándar binario, diseñado para la confiabilidad, seguridad, aprovisionamiento e interoperabilidad (Naik, 2017).

Las características del protocolo según (Pérez Leones, 2019), (Pohl, Kubela, Bosse, & Turowski, 2018) son:

- Está orientado a mensajería

- Puerto predeterminado 5671 (TLS/SSL) y 5672
- Con arquitectura Cliente/Bróker o Cliente/Servidor
- Posee cola de mensajes o (queuing)
- Tamaño del mensaje negociable e indefinido
- Admite diferentes protocolos de transporte, pero utiliza como protocolo de transporte predeterminado TCP
- Enrutamiento (punto a punto y publicación /suscripción)
- Proporciona seguridad utilizando TLS/SSL para el cifrado y SASL para la autenticación
- El intercambio de mensajes es asíncrono y garantiza la entrega de los mensajes a través del intercambio de mensajes de control
- Permite la mensajería transaccional.
- Cuenta con tres niveles de calidad de servicio (QoS) (como máximo una vez, al menos una vez, exactamente una vez), es decir, QoS 0 no requiere confirmación del receptor al transmisor, QoS 1 requiere confirmación de recepción del mensaje y QoS 2 garantiza que el mensaje se entrega solo una vez sin repeticiones.
- Define un protocolo de transporte (protocolo a nivel de conexión) como una capa semántica de alto nivel para que distintas implementaciones AMQP puedan interoperar entre sí.

6.3.1 Estructura del protocolo AMQP (Advanced Message Queuing Protocol)

En (Pérez Leones, 2019) refiere que AMQP posee un modelo en capas como se aprecia en la figura 23, las cuales se describen brevemente a continuación:

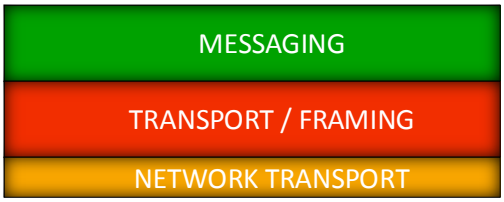


Figura 26. Modelo en capas de AMQP
Adaptado de la figura 2.17 de (Pérez Leones, 2019)

- Network Transport o de transporte de red: En esta capa es donde se establece la conexión con cualquier protocolo de transporte como TCP o SCTP.
- Transport/Framing o de transporte/entramado: Aquí se establece la conexión AMQP especificando el comportamiento de conexión y la seguridad entre pares (nodos y contenedores). Los nodos AMQP pueden ser: productores, consumidores y colas; mientras que, los contenedores AMQP pueden ser brókers (Intermediarios) y aplicaciones cliente. Además, dentro de un contenedor existen varios nodos.

Para establecer una comunicación entre nodos de diferentes contenedores, es necesario establecer una conexión AMQP, que consiste en una comunicación dúplex completa con una secuencia ordenada de tramas, cuyo tamaño máximo de trama es negociado. Está conexión AMQP, se divide en varios canales unidireccionales independientes.

Después de la conexión, se crea una sesión AMQP entre dos pares, enlazando dos canales unidireccionales para formar una comunicación bidireccional y secuencial. En efecto, una sola conexión puede tener múltiples e independientes sesiones activas simultáneamente y, por tanto, varios canales AMQP al mismo tiempo, como se aprecia en la figura 27.

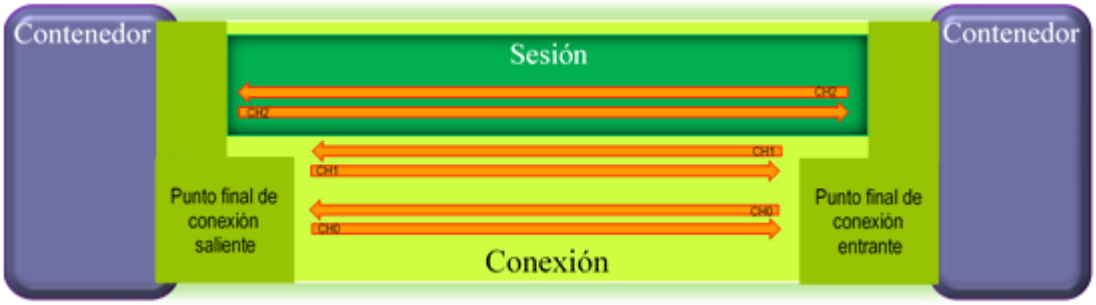


Figura 27. Conexión AMQP
Adaptado de (Microsoft Azure, 2020)

- Messaging o de mensajería: Es donde se especifica un uso estandarizado de los mensajes para proporcionar capacidades de mensajería interoperables.

6.3.2 Arquitectura y funcionamiento AMQP (Advanced Message Queuing Protocol)

La arquitectura del protocolo AMQP, se aprecia en la figura 28 y está compuesta de las siguientes entidades (Sciarrone & Sauro, 2017) (Pérez Leones, 2019)

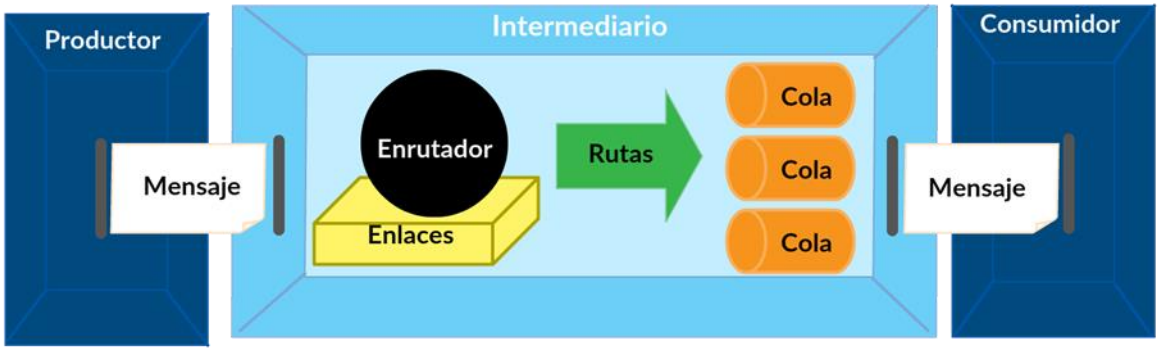


Figura 28. Arquitectura protocolo AMQP
Adaptado de la figura 2.7 (Sciarrone & Sauro, 2017)

- Productor: Es una aplicación cliente que publica mensajes AMQP en el intermediario
- Consumidor: Es una aplicación cliente que solicita mensajes de una cola de mensajes

- Intermediario: Es un servidor al que se conectan productores y consumidores para enviar y recibir mensajes.
- Enrutador: Es una entidad que recibe los mensajes del intermediario encaminándolos a las colas correspondientes. Existen 4 tipos de enrutadores y se diferencian sólo en el algoritmo utilizado para determinar qué cola debe recibir los mensajes, estos son:

✓ Enrutador Directo: Los mensajes se envían a las colas con una clave de enrutamiento igual a la utilizada para el enlace, como en la figura 29. Se usan principalmente para unidifusión, pero también se pueden usar con multidifusión.

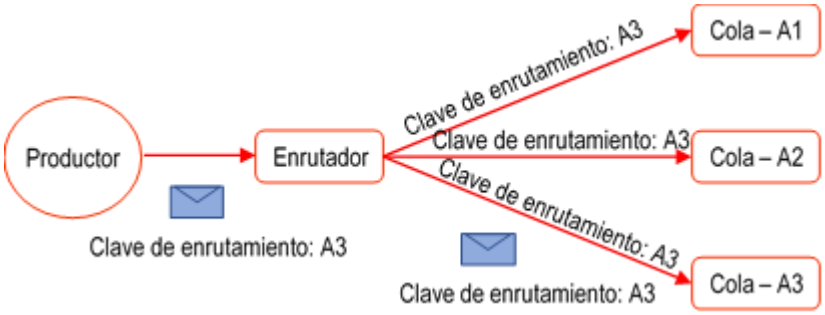


Figura 29. Enrutador Directo
Adaptado de (Mesa, 2019)

✓ Enrutador por despedida: Envía todos los mensajes a cada cola que esté vinculada a ese enrutador, como en la figura 30. Es ideal para tráfico broadcast o difusión masiva.

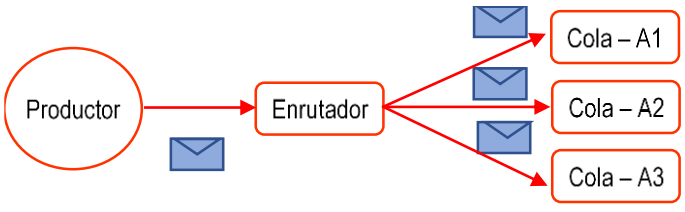


Figura 30. Enrutador por despedida
Adaptado de (Mesa, 2019)

- ✓ Enrutador por tema: Los mensajes son enviados a las colas cuando el patrón definido por la clave de enlace coincide con la clave de enrutamiento del mensaje, como en la figura 31. Se usa para comunicación multidifusión.

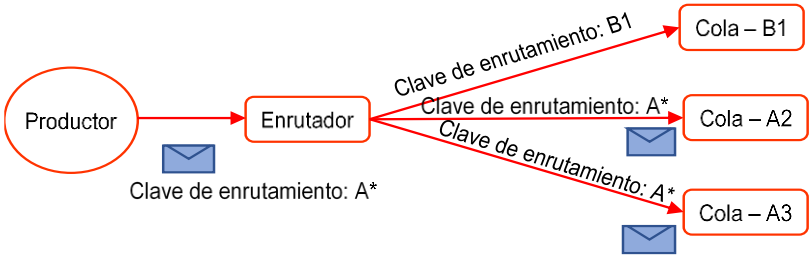


Figura 31. Enrutador por tema
Adaptado de (Mesa, 2019)

- ✓ Enrutador por cabecera: Los mensajes son enrutados a las Colas que cumplan con la información incluida en la cabecera del mensaje. Se usa para comunicación basada en metadatos.
  - Enlaces: Es la entidad que determina las reglas utilizadas por el enrutador para encaminar un mensaje a una cola.
  - Colas: Es la entidad responsable de almacenar y distribuir mensajes para el consumidor que se suscriba a ella. Tienen la capacidad de almacenar hasta millones de mensajes al mismo tiempo.

Para establecer una comunicación AMQP, el Productor envía o publica un mensaje al Enrutador, quien a su vez reenvía el mensaje dependiendo de reglas determinadas por los Enlaces a una o múltiples Colas asociadas a los Consumidores (Povedano Molina, 2014).

6.3.3 Trama AMQP (Advanced Message Queuing Protocol)

La frame o trama es la unidad básica en AMQP y está dividida según (ISO/IEC 19464:2014, 2014) en tres áreas, como en la figura 32, las cuales son:

- ***Frame header:*** Es una estructura de tamaño fijo (8 bytes), con información obligatoria necesaria para analizar el resto de la trama.
- ***Extended header:*** Es un área de ancho variable, definido para una futura expansión. El tratamiento de esta área depende del tipo de trama.
- ***Frame body:*** El cuerpo de la trama es una secuencia de bytes de ancho variable cuyo formato depende del tipo de trama.

En la trama AMQP de la figura 32 los bytes 0-3 corresponden al campo SIZE (tamaño de la trama), el byte 4 es el DOFT (desplazamiento de datos) muestra la posición del cuerpo dentro de la trama, el byte 5 es el código TYPE o código de tipo que indica el formato y el propósito de la trama; por ejemplo, un código de tipo 0x00 indica que la trama es una trama AMQP, un código de tipo 0x01 indica que es una trama SASL y los bytes 6 y 7 contienen el número de canal.

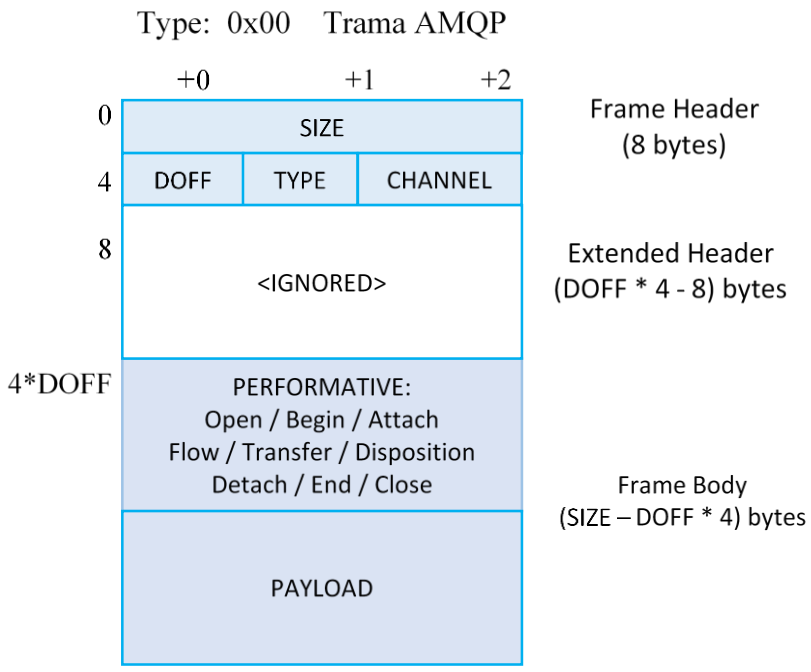


Figura 32. Formato trama AMQP

Adaptado de la Figura 2.16 (ISO/IEC 19464:2014, 2014)

El cuerpo de la trama está compuesto por el campo performativo, y el campo payload o carga útil. Payload es el conjunto de datos a transmitir y el performativo es un cuerpo de trama utilizado para la transferencia de mensajes entre dos pares. En (ISO/IEC 19464:2014, 2014) refiere 9 tipos de performativos los cuales se aprecian en la tabla 15.

Tabla 15.

Performativos AMQP

Tipo	Descripción
Open	Negocia parámetros de conexión entre el cliente y el bróker.
Begin	Comienza una sesión en un canal.
Attach	Adjunta un enlace a una sesión.
Flow	Actualiza el estado del enlace.
Transfer	Transfiere un mensaje.
Disposition	Informa a los pares remotos de los cambios de estado de entrega.
Detach	Sirve para eliminar el enlace.
End	Finaliza la sesión.

6.3.4 Comunicación AMQP (Advanced Message Queuing Protocol)

El proceso de comunicación entre pares inicia con una conexión, después se establece una sesión dentro de esta conexión y finalmente sucede el intercambio de mensajes. Los pasos implicados en este proceso se describen en la tabla 16.

Tabla 16.

Proceso de comunicación AMQP

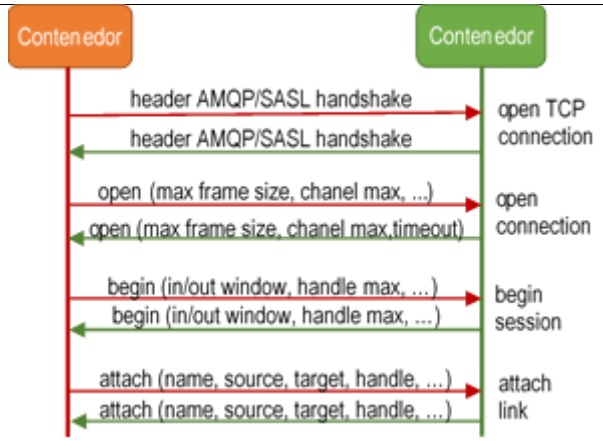
Tipo	Ejemplo
Comunicación open: La comunicación comienza cuando se establece la conexión TCP a través de un handshake (apretón de manos) entre AMQP/SASL, después se abre una conexión al intercambiar el performativo **open** AMQP, allí se precisa el tamaño máximo de trama (control de flujo), el número máximo de canales, entre otros. Al interior de esta conexión se inicia una sesión utilizando **begin** y se fija el tamaño de la ventana (número de tramas para el control de flujo). Finalmente se utiliza **attach** para adjuntar un enlace (Pérez Leones, 2019).	

Figura 33. Comunicación open
Adaptación figura 2.20 (Pérez Leones, 2019)

Comunicación send: Después de adjuntar el enlace, el receptor envía un performativo **flow** al productor para definir el número de créditos, es decir, el número de mensajes que puede recibir (control de flujo). Para enviar datos el productor utiliza **transfer** y el receptor responde con **disposition** si la QoS está en nivel uno (al menos una vez) (Pérez Leones, 2019).

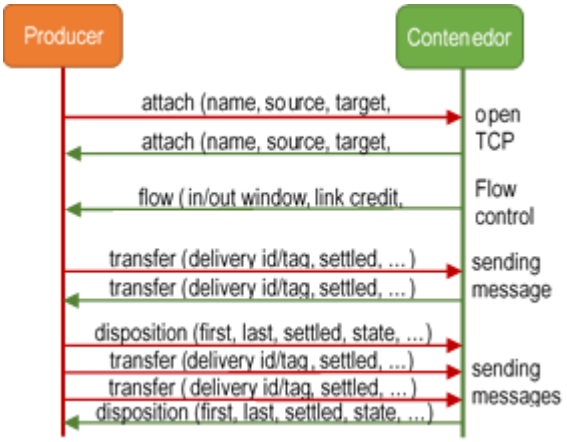


Figura 34. Comunicación send
Adaptación figura 2.21 (Pérez Leones, 2019)

Comunicación receive: Esta comunicación se dirige en sentido contrario al envío. El receptor envía un performativo **flow** al consumidor y a través de un control de flujo basado en créditos se determina cuantos mensajes puede recibir antes de gestionarlos. Al enviar datos el productor con **transfer**, el receptor responde con **disposition** si el nivel de QoS es mayor que cero (Pérez Leones, 2019).

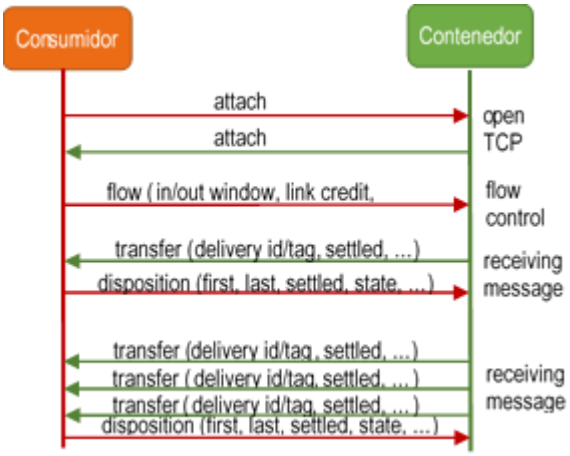


Figura 35. Comunicación receive
Adaptación figura 2.22 (Pérez Leones, 2019)

Comunicación close: Para cerrar una comunicación se eliminan todos los enlaces activos con el performativo **detach,** luego, con **end** se finaliza la sesión **y con close** se cierra la conexión, por último, se termina la conexión de red (Pérez Leones, 2019).

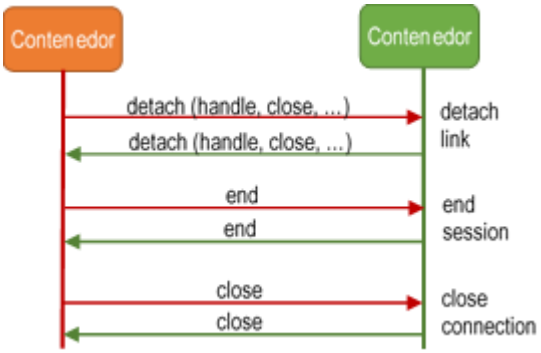


Figura 36. Comunicación close
Adaptación figura 2.23 (Pérez Leones, 2019)

6.3.5 Formato del mensaje AMQP (Advanced Message Queuing Protocol)

Se definen dos tipos de mensaje: Bare Massages o mensajes desnudos que es el mensaje tal como lo suministró el remitente y Annotated Message o mensaje anotado que consiste en el mensaje desnudo más secciones para anotaciones en la cabecera y cola del mensaje desnudo, tal como como se ve en el receptor (ISO/IEC 19464:2014, 2014). El formato del mensaje se aprecia en la figura 37 y consta de las siguientes secciones:

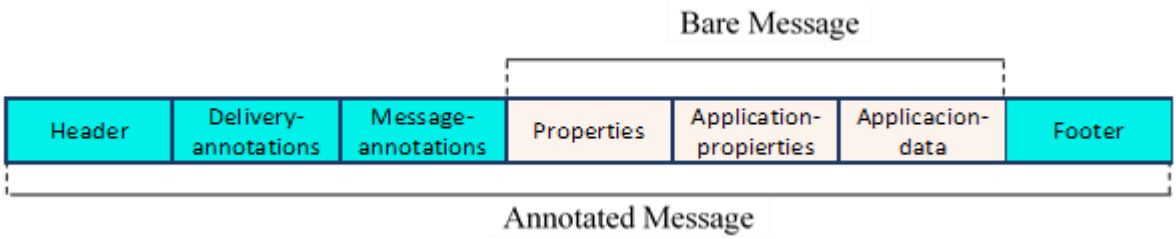


Figura 37. Formato del mensaje AMQP
Adaptado de (ISO/IEC 19464:2014, 2014)

- ***Header:*** Contiene detalles de entrega sobre la transferencia de un mensaje a través de la red AMQP.
- ***Delivery-annotations:*** Especifica propiedades no estándar de entrega, en la cabecera del mensaje.
- ***Message-annotations:*** Se usa para las propiedades del mensaje que están dirigidas a la infraestructura y deben propagarse en cada paso de entrega.
- ***Properties:*** Se utiliza para un conjunto definido de propiedades estándar del mensaje.
- ***Application-properties:*** Es parte del mensaje básico utilizado para los datos estructurados de la aplicación
- ***Body:*** Consta de una de las siguientes tres opciones: una o más secciones de datos, una o más secciones de secuencia AMQP o una sola sección de valor AMQP.

- ***Footer:*** Transporta pies de página para un mensaje.

6.3.6 Control de flujo

El protocolo AMQP proporciona control de flujo en diferentes niveles. Un primer nivel de control de flujo ocurre, durante el proceso de conexión, cuando los pares negocian un tamaño de trama máximo y definen el tamaño de cada trama individual a intercambiar (Patierno, 2018).

Otro nivel de control de flujo es durante la sesión y está basado en ventanas, es decir, cuando se crea una sesión, cada parte define el número de tramas a aceptar en su ventana de recepción, dando lugar al intercambio de tramas entre las partes; sin embargo, este intercambio de tramas se detiene al llenarse la ventana (Patierno, 2018).

El último nivel de control de flujo está en el proceso de mensajería, donde cada enlace se basa en el "crédito de enlace" que es un contador de mensajes establecido por el receptor utilizando un performativo de "flujo" para cada mensaje entrante. Cada entrega de mensaje disminuye el crédito de enlace y cuando se agota el crédito las entregas se detienen (Patierno, 2018).

6.3.7 Características extendidas

El protocolo AMQP posee extensiones que introducen características adicionales al protocolo para mejorar sus capacidades, las cuales son:

- **Advanced message queuing protocol (AMQP) WebSocket Binding (WSB) version 1.0:** Esta extensión describe cómo se puede usar el protocolo WebSocket como transporte para el tráfico del protocolo AMQP 1.0, es aplicable en escenarios como firewall y mensajería basada en navegador web (Fallows, Ingham, & Godfrey, 2016)

- **Using the AMQP anonymous terminus for message routing version 1.0:** Esta especificación define un mecanismo mediante el cual un único enlace saliente se puede utilizar para transferir mensajes que luego se enrutan utilizando la dirección que se encuentra en su campo "a" (Godfrey, 2018b).
- **Advanced message queuing protocol (AMQP) enforcing connection uniqueness version 1.0:** Esta extensión se define un mecanismo para asegurar que solo exista una conexión AMQP abierta entre dos procesos que se comunican usando AMQP V1.0 (Godfrey, 2018a).

6.3.8 Implementaciones

Las aplicaciones AMQP se pueden implementar a partir de un proceso de construcción mediante APIs o utilizando implementaciones genéricas diseñadas para varias plataformas. En la tabla 17 se mencionan algunas de las librerías más utilizadas y sus características como: lenguaje de programación, manejo de mensajes, seguridad, plataforma de destino e interoperabilidad.

Tabla 17.

Implementaciones AMQP

Implementación	Descripción
RabbitMQ	Es un software de negociación de mensajes, en Erlang, desarrollado por la compañía Pivotal con licencia pública de Mozilla. Tiene librerías para servidor y cliente en la mayoría de los sistemas operativos y plataformas. Admite AMQP 0-9-1 y AMQP 1.0 a través de un complemento experimental. Admite colas de mensajes, confirmación de entrega de mensajes, enrutamiento flexible a colas, clusterización, federación y tolerancia a fallas. Para la seguridad admite TLS, LDAP y SASL. (RabbitMQ by Pivotal, 2007)
Qpid Broker-J	Es un servidor de mensajería en Java, desarrollado por la fundación Apache con licencia Apache 2.0. Admite todas las versiones de AMQP, permite la

	administración a través de una API, AMQP Management y consola web. Maneja control de flujo del productor. Las opciones de almacenamiento de mensajes incluyen Derby, SQL y BDB y manejo de mensajes no entregables. Admite la extensión AMQP sobre WebSocket. La autenticación admite certificados LDAP, Kerberos, O-AUTH2 y SSL. (Apache Qpid™, 2015a) (Apache Qpid, 2020)
Qpid C++ Broker	Es un servidor de mensajería en C++, desarrollado por la fundación Apache con licencia Apache 2.0. Admite AMQP 0-10 y 1.0. Maneja control de flujo del productor. Permite la replicación de colas. Con manejo de mensajes no entregables. Incluye las opciones de almacenamiento de mensajes. Permite la autenticación SASL. (Apache Qpid™, 2015b)
Qpid Proton	Es un kit de herramientas de mensajería AMQP, desarrollado por la fundación Apache con licencia Apache 2.0. Se puede utilizar en brókers, bibliotecas de clientes, enrutadores, servidores proxy etc. Tiene control total de la semántica del protocolo AMQP 1.0. Se integra con el ecosistema AMQP 1.0 desde cualquier plataforma, entorno o idioma. Admite la mensajería punto a punto o con bróker. Para la seguridad utiliza SSL y SASL. (Apache Qpid™, 2015d)
Qpid JMS	Es un cliente AMQP 1.0 en JMS 2.0 (Java Message Service), desarrollado por la fundación Apache con licencia Apache 2.0. Maneja control de flujo del productor, tolerancia a fallas y para la seguridad usa SSL y SASL. (Apache Qpid™, 2015c)
Apache ActiveMQ	Es un servidor de mensajería en Java junto con un cliente en JMS, desarrollado por la fundación Apache con licencia Apache 2.0. Admite AMQP 1.0. Soporta numerosos protocolos y plataformas. Tiene alta disponibilidad usando almacenamiento compartido, ofrece clusterización, con una red de brókers para distribuir carga y opciones de KahaDB y JDBC para almacenamiento. (Apache ActiveMQ, 2019)
Azure Service Bus	Es un software de mensajería en la nube, desarrollada por Microsoft con licencia Microsoft CLUF (licencia de usuario final). Admite AMQP 1.0. Permite el procesamiento por lotes o sesiones. Admite transacciones, colas de mensajes fallidos, control temporal, enrutamiento, filtrado y detección de duplicados de mensajes. Para seguridad admite SAS (firmas de acceso compartido) y RBAC (Control de acceso basado en rol). (Microsft Azure, 2020)

6.3.9 Casos de aplicación

El protocolo AMQP es utilizado en varias áreas de aplicación debido a su robustez, fiabilidad e interoperabilidad, en la tabla 18 se mencionan algunos casos de uso que buscan estos requisitos en diferentes entornos.

Tabla 18.

Casos de aplicación AMQP

Área	Ámbito de Aplicación
HOGARES INTELIGENTES	AMQP es usado en el hogar para automatizar actividades como: • La medición de fugas de gas en los equipos de consumo. (Pérez Leones, 2019) • Monitoreo y control de dispositivos en el hogar. (Adiono, Manangkalangi, Muttaqin, Harimurti, & Adijarto, 2017) (Adiono, Marthensa, et al., 2017) (Sreeraj & Kumar, 2018)
AMBIENTE INTELIGENTE	AMQP es usado en el medio ambiente para automatizar actividades como: • Monitoreo ambiental en el mar. (Meera & Rao, 2018) • Monitoreo meteorológico en una zona rural en condiciones climáticas adversas (Kostromina, Siemens, & Babich, 2018) • Monitoreo del espectro electromagnético del campus en una Universidad (Lv, Meng, & Zhang, 2014)
SALUD	AMQP es usado en la salud para automatizar actividades como: • Monitoreo remoto de pacientes en tiempo real. (Liang & Chen, 2018) (Krishna & Sasikala, 2019) • Diagnóstico médico de enfermedades neurodegenerativas a partir de la EMT (estimulación magnética transcraneal). (Depari et al., 2019) • Monitoreo remoto de pacientes a partir de PHD (dispositivos de salud personal). (Bellagente et al., 2018) • Sistema de audio para la monitorización remota del estado de salud de los pacientes (Rubio Conde, Villarán Molina, & García Valls, 2017) • Monitoreo de la rehabilitación respiratoria de pacientes con EPOC (enfermedad pulmonar obstructiva crónica) en interiores

	y en movilidad.(Talaminos Barroso, Estudillo Valderrama, Roa, Reina Tosina, & Ortega Ruiz, 2016)
CONSERVACION DE LA ENERGIA	AMQP es usado en la conservación de la energía para automatizar actividades como: • Monitoreo y gestión del consumo de energía eléctrica (Smart grid). (Albano, Lino Ferreira, Pinho, & Alkhawaja Rahman, 2015)
CIUDAD INTELIGENTE	AMQP es usado en la ciudad para automatizar actividades como: • Iluminación inteligente de un parque. (Merlino et al., 2015)
INDUSTRIA	AMQP es usado en la industria para automatizar actividades como: • Inspección de productos para control de calidad. (Llamuca, Garcia, Naranjo, & Garcia, 2019) • Comunicación entre dispositivos en una fábrica inteligente. (Bezerra, Aschoff, Szabo, & Sadok, 2018)

6.3.10 Otros aspectos

6.3.10.1 Seguridad La seguridad en el protocolo AMQP se establece a nivel de la capa de transporte, en la conexión TCP, a través del protocolo SSL/TLS para encriptar el canal de comunicación. Y las tres formas de establecer SSL/TLS en AMQP son: TLS dentro de AMQP (puerto 5672), AMQP dentro de TLS (puerto 5671) y túnel de WebSocket. Mientras que, para la autenticación se utiliza el protocolo SASL con mecanismos como: ANONYMOUS (sin autenticación), PLAIN (nombre de usuario y contraseña), etc. (Patierno, 2018)

La falta de seguridad en la transmisión de datos entre dispositivos que utilizan IoT, ha incentivado a varios investigadores a presentar métodos, políticas y mecanismos de seguridad orientados a proteger estas redes junto con el protocolo AMQP como los siguientes:

En (Adiono, Manangkalangi, et al., 2017) garantizan la seguridad de los datos, al cifrarlos utilizando los algoritmos RSA y AES, para evitar que los piratas informáticos hackeen el sistema

durante la comunicación AMQP. Aunque AES es más seguro que RSA, el sistema necesita RSA porque tiene un par de claves estáticas conocidas por ambas partes, mientras que AES proporciona una clave dinámica que siempre cambia cuando el usuario inicia sesión y cuando inicia sesión dentro de las 24 horas.

En (Verma, 2017) aborda la seguridad de los datos con una tecnología Hadoop, la cual consiste en recopilar los datos y mediante un modelo computacional realizar un proceso de curación de datos para detectar fraudes, defectos, entre otros, antes de entregar los datos a los usuarios.

En (Ammar, Russello, & Crispo, 2018) examina la seguridad de varias plataformas para IoT, por ejemplo, Azure IoT Suite de Microsoft con AMQP integrando la seguridad en: los dispositivos, la conexión y la nube, para esto utiliza: autenticación mutua, cifrado con TLS, autenticación con el certificado X.509, seguridad en la nube con políticas de control de acceso y cifrado con SSL/TLS para garantizar la integridad y confidencialidad de los datos.

6.4 Protocolo WebSocket

WebSocket es un protocolo de comunicación bidireccional full dúplex y con estado, que fue normalizado por la IETF y se especifica en la RFC 6455 (Fette & Melnikov, 2011)**.** EL protocolo WebSocket fue diseñado para dar solución a problemas como la alta sobrecarga de comunicación de internet que resultaban del uso de tecnologías de comunicación bidireccional que utilizaba HTTP (Loreto, Saint-Andre, Salsano, & Wilkins, 2011), (Soares de Souza et al., 2018). Las principales características del protocolo WebSocket son:

- ✓ Presenta arquitectura cliente/servidor.

- ✓ Trabaja sobre TCP.
- ✓ Funciona sobre los puertos HTTP 80 y 443.
- ✓ Es un protocolo basado en servicios web, pero se puede implementar en cualquier otro tipo de aplicación de arquitectura cliente/servidor.
- ✓ Puede ser escrito en cualquier lenguaje servidor.
- ✓ Admite proxy e intermediarios HTTP.
- ✓ Utiliza certificados TLS/SSL para garantizar la seguridad de la información.

El protocolo WebSocket combinado a la API que define la W3C permite a las páginas web la creación y administración de una conexión bidireccional a un servidor con el fin de enviar y recibir datos en una única conexión TCP. Esta API de WebSocket está siendo normalizada por el consorcio World Wide Web Consortium (W3C) (Kaazing, s/f), (W3C, 2012) . En la actualidad, la mayoría de los navegadores ya admiten WebSocket, como se referencia en (Deveria, 2018).

6.4.1 Estructura y funcionamiento

El protocolo WebSocket presenta tres partes: establecimiento de la comunicación, la transferencia de datos y el cierre de la conexión, como se puede visualizar en la figura 38.

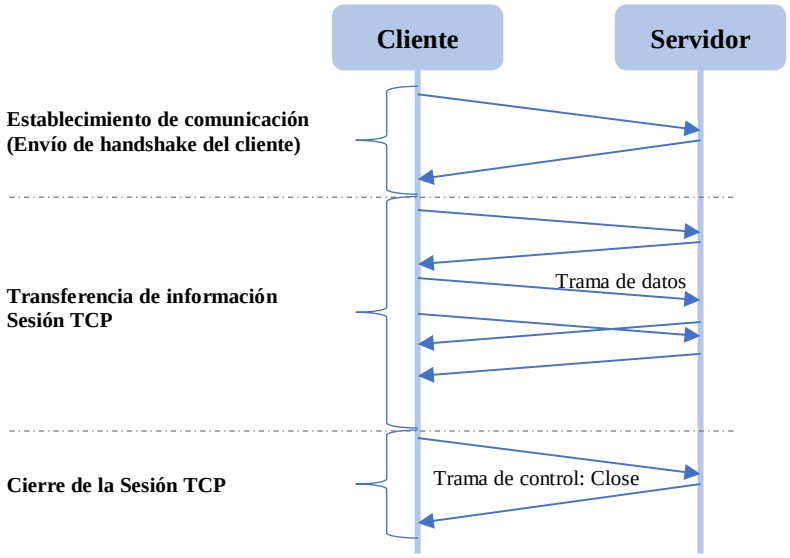


Figura 38. Partes del Protocolo WebSocket

Adaptado de figura1 de (Huawei Cloud, s/f)

6.4.1.1 Establecimiento de la comunicación Es la primera etapa del protocolo WebSocket, en la que el cliente abre una conexión TCP (estado de la conexión CONNECTING) y envía un handshake de apertura. El handshake de apertura del cliente consta de una solicitud Upgrade HTTP, acompañado de una lista de campos obligatorios y opcionales que juntos cumplen una serie de requisitos que se definen en la sesión 4 de (Fette & Melnikov, 2011). A continuación, se visualiza en la figura 39, el código del handshake de apertura del cliente:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Figura 39. Handshake del Cliente

Adaptado de la sección 1.2 de (Fette & Melnikov, 2011)

El cliente debe proporcionar los campos /host/, /port/, /resource name/ y /secure/, que componen el URI de WebSocket. Para WebSocket se definen dos URI, como son:

ws-URI = "ws:" "//" host [":" port] path ["?" query]

wss-URI = "wss:" "//" host [":" port] path ["?" query]

En donde, el componente /port/ es opcional; "ws" es el valor predeterminado para el puerto 80, mientras que para "wss" es el puerto 443, como se menciona en la sección 3 de (Fette & Melnikov, 2011).

A continuación, en la tabla 19, se describen brevemente los estados de una conexión WebSocket:

Tabla 19.

Estados de la conexión WebSocket

Constante	Valor	Descripción
CONNECTING	0	Estado inicial de una conexión. Aún no está abierta.
OPEN	1	La conexión está abierta y lista para comenzar a enviar y recibir datos.
CLOSING	2	La conexión está siendo cerrada.
CLOSED	3	La conexión está cerrada o no puede ser abierta.

Nota: Adaptada de (Fette & Melnikov, 2011)

Si el servidor soporta el protocolo WebSocket, debe analizar el handshake de apertura del cliente para obtener la información necesaria para generar el handshake de apertura del servidor. En caso de que el servidor no pueda procesar la solicitud del cliente retornara una respuesta HTTP con un código de error apropiado (como 400 Bad Request).

El handshake de apertura del servidor es mucho más simple que el del cliente y se visualiza en la figura 40, a continuación:

```
HTTP/1.1 101 Switching Protocols
    Upgrade: websocket
    Connection: Upgrade
    Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Figura 40. Handshake del Servidor
Adaptada de la sección 1.2 de (Fette & Melnikov, 2011)

Donde, el código de estado 101 indica que el protocolo de enlace WebSocket se ha completado, los campos de encabezado |***Upgrade***| y |***Connection***| completan la actualización HTTP y el campo de encabezado |***Sec-WebSocket-Accept***| indica si el servidor está dispuesto a aceptar la conexión. Si está presente, este campo de encabezado debe incluir un hash del nonce del cliente enviado en |***Sec-WebSocket-Key***| junto con un GUID predefinido. Cualquier otro valor no debe interpretarse como una aceptación de la conexión por parte del servidor.

Si el servidor concluye el análisis de la información del handshake de apretura del cliente sin abortar la solicitud, el servidor envía un handshake de apertura al cliente, indicando que se ha establecido la conexión de WebSocket y esta pasa a estado OPEN. En la figura 41, se aprecia el proceso de establecimiento de la comunicación.

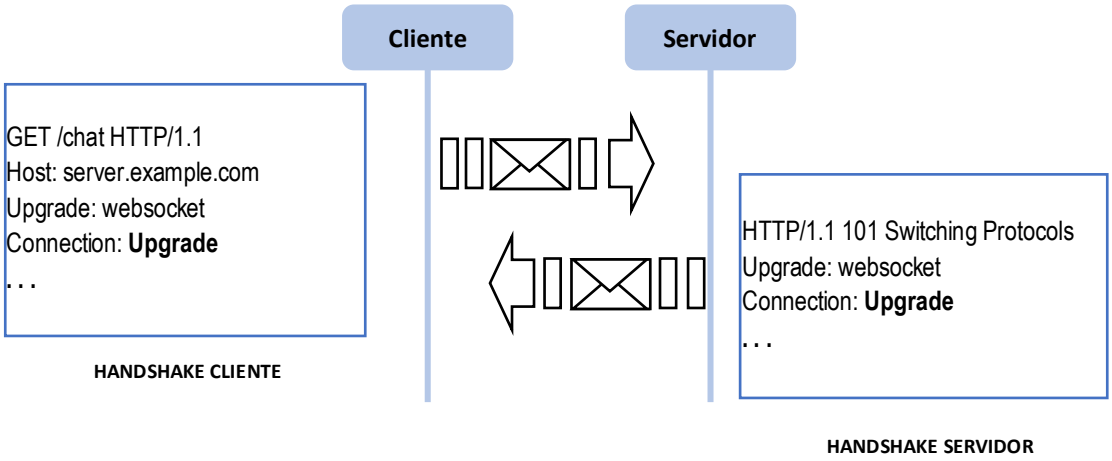


Figura 41. Establecimiento de la conexión WebSocket
Adaptado de (Fette & Melnikov, 2011)

6.4.1.2 Transferencia de información Una vez que el cliente y el servidor han enviado su handshake, y la conexión WebSocket es establecida en estado OPEN, se da inicio a la segunda parte, la transferencia de datos, como se ilustra en la figura 42.

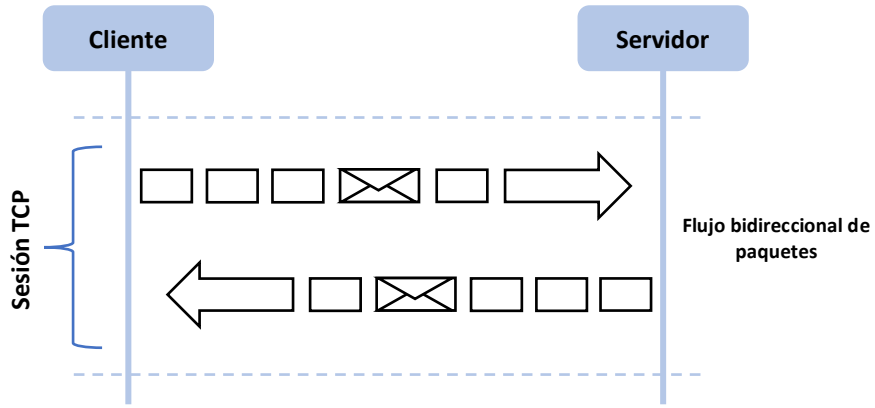


Figura 42. Transferencia de datos WebSocket
Adaptada de (Fette & Melnikov, 2011)

En el protocolo WebSocket, la transferencia de datos es a través de una secuencia de tramas sobre una sesión TCP, en el que cada lado puede enviar información, en cualquier momento.

Para enviar un mensaje, un punto final encapsula la información en una o más tramas WebSocket, en donde cada trama tiene un tipo de dato que debe coincidir con el resto de las tramas del mismo mensaje. En el protocolo WebSocket para enviar y recibir se deben cumplir unos requisitos los cuales se definen en la sección 6.1 y 6.2 de (Fette & Melnikov, 2011).

6.4.1.3 Cierre de la conexión Para iniciar el cierre de una conexión WebSocket, un punto final debe enviar una trama de control de cierre, indicando el código de estado y opcionalmente un motivo de cierre. La conexión pasa a estado CLOSING (ver tabla 1). Normalmente el servidor inicia el cierre de la conexión WebSocket pero en casos anormales el cliente puede iniciar también el cierre de la conexión WebSocket, como se especifica en la sección 7 de (Fette & Melnikov, 2011).

6.4.2 Trama WebSocket

En WebSocket, tanto el cliente como el servidor puede decidir enviar un mensaje en cualquier momento y todas las tramas siguen el mismo formato. La sección 5 de la especificación se describe esto en detalle (Fette & Melnikov, 2011). En la figura 43, se ilustra el formato de la trama WebSocket.

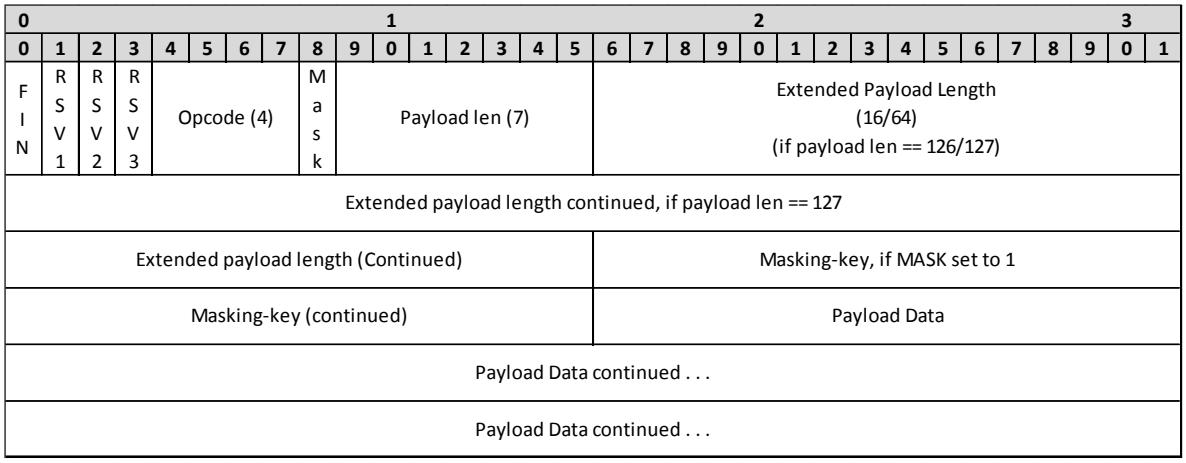


Figura 43. Formato de trama WebSocket

Adaptada de la sección 5.2 de (Fette & Melnikov, 2011)

A continuación, se describen los campos de la trama WebSocket:

- ***FIN:*** 1 bit. Indica si la trama es la última de la serie.
- ***RSV1, RSV2 y RSV3***: 1 bit para cada uno. Son para uso de extensiones.
- ***Opcode:*** 4 bits. Define el tipo de datos que contiene la trama. Ver sección 7.2.5. de (Fette & Melnikov, 2011).
- ***Mask:*** 1 bit. Indica si el campo Payload data está codificado.
- ***Payload length:*** 7 bits. define la longitud del campo Payload data en bytes.
- ***Masking-key:*** 16 bits. Contiene la máscara utilizada para enmascarar el campo Payload data y solo está presente cuando el campo Mask es igual a 1.
- ***Payload data:*** Contiene el mensaje.

6.4.3 Fragmentación

El protocolo WebSocket cuenta con un mecanismo de fragmentación y en la sección 5.4 de (Fette & Melnikov, 2011), define las reglas que se aplican para su funcionamiento. La finalidad

del mecanismo es permitir enviar un mensaje de tamaño desconocido. Las tramas de control no deben estar fragmentadas y deben tener un tamaño de carga útil de 125 bytes.

6.4.4 Tramas de control

Las tramas de control se utilizan para comunicar el estado de la conexión WebSocket. Se identifican mediante códigos de operación y se definen en la tabla 20.

Tabla 20.

Tipos de tramas WebSocket según código de operación

Códigos de Operación	Tipo de trama
%x0	Trama de continuación
%x1	Trama Texto
%x2	Trama Binaria
%x3-%x7	Reservados para futuras tramas sin control
%x8	Trama de cierre de conexión
%x9	Trama Ping
%xA	Trama Pong
%xB-0xF	Reservados para futuras tramas control

Nota: Adaptado de (Fette & Melnikov, 2011)

6.4.5 Extensibilidad

El protocolo está diseñado para permitir extensiones, que agregarán capacidades al protocolo. En la tabla 20 se definen los códigos de operación reservados para futuras extensiones(Fette & Melnikov, 2011).

6.4.6 Características extendidas

Existen extensiones que agregan características que aumentan las capacidades del protocolo WebSocket. A continuación, se presentan algunas de estas:

- **RFC 7692 Compression extensions for WebSocket:** Este marco define un método para aplicar un algoritmo de compresión al contenido de los mensajes de WebSocket (Yoshino, 2015).
- **RFC 7936 Clarifying registry procedures for the WebSocket subprotocol name registry:** Describe una actualización de la política de registro en la sección 11.5 de RFC 6455, no es un protocolo (Hardie, 2016).
- **RFC 7977 The WebSocket protocol as a transport for the message session relay protocol (MSRP):** Especifica un nuevo subprotocolo de WebSocket como un mecanismo de transporte confiable entre clientes y retransmisores de Protocolo de retransmisión de sesión de mensaje (MSRP) para permitir el uso de MSRP en nuevos escenarios (Dunkley, Llewellyn, Pascual, Salgueiro, & Ravindranath, 2016).
- **RFC 8307 Well-Known URIs for the WebSocket protocol:** Actualiza la RFC 6455 para extender el uso de RFC 5785 URI conocidos (/well-known), a los esquemas de URI "ws" y "wss" (Borman, 2018).
- **RFC 8441 bootstrapping WebSockets with HTTP/2:** Define un mecanismo para ejecutar el Protocolo WebSocket en una sola secuencia de una conexión HTTP/2 (McManus, 2018).
- **RFC 8323 CoAP (Constrained application protocol) over TCP, TLS, and WebSocket:** Describe los cambios necesarios para usar CoAP sobre TCP, TLS y transportes WebSocket. La sección 4 de (C. Bormann et al., 2018), define varias

configuraciones que se pueden usar para implementar protocolo CoAP sobre el protocolo WebSocket, como recuperar un recurso ubicado en un servidor CoAP a través de una conexión WebSocket.

6.4.7 Implementaciones

Según (Deveria, 2018), WebSocket es soportado por muchos navegadores web existentes, como Google Chrome, Firefox y Edge, entre otros. En la tabla 21 se describen algunas implementaciones del protocolo WebSocket.

Tabla 21.

Implementaciones WebSocket

Implementación	Descripción
Gorilla WebSocket	Es una implementación del protocolo WebSocket escrita en lenguaje Go. Permite fragmentación, Pings y Pongs, Opcodes, Handshake de cierre, entre otras características definidas en RFC6455. (Silverlock, 2019)
Websocketd	Es una herramienta de línea de comandos que permitirá a otros programas de interfaz de línea de comandos acceder a través de un WebSocket. Permite crear un servidor WebSocket si lee STDIN y escriba en STDOUT. No se necesitan bibliotecas de redes. (Sergeyev, 2019)
WebSocket ++ (0.8.1)	Es una biblioteca de cliente/servidor WebSocket, escrita en lenguaje C++. Implementa la RFC6455, permite integrar la funcionalidad del cliente y servidor WebSocket en programas C ++. (Zaphoyd Studios, s/f) (Thorson, 2020)
Libwebsockets	Es una biblioteca escrita en lenguaje C que proporciona cliente y servidor para WebSocket y otros protocolos. Admite SSL/TLS para su seguridad. Maneja licencia MIT. (w3.css, 2020)
EM-Websocket	Es una implementación del protocolo WebSocket para EventMachine. Escrita en lenguaje Ruby. Responde ante determinados eventos, como puede ser la llegada de un mensaje o el surgimiento de un error (ws.onmessage y ws.onerror). Maneja Licencia MIT (Wikipedia, 2018) (Grigorik, 2018)
javax.websocket	Paquete de Java que contiene todas las API de WebSocket comunes al lado del cliente y del servidor. Maneja licencia BCL. (Oracle, 2015)
SocketRocket	Es una biblioteca de cliente WebSocket. Escrita en Objective-C. Se ajusta a la RFC 6455, admite wss ya que la conexión se basa en CFStream (enlazado

	a NSStream). Compatible con iOS, macOS y tvOS. Maneja licencia BSD. (Hammond & Lutsenko, 2018)
Elephant.io	Es un cliente WebSocket básico escrito en PHP. Su objetivo es facilitar las comunicaciones entre su aplicación PHP y un servidor en tiempo real. Maneja Licencia MIT. (Aires, 2017)

6.4.8 Casos de aplicación

El protocolo WebSocket fue diseñado originalmente para implementarse en navegadores web y servidores web, pero se ha extendido para usarse en cualquier aplicación cliente o servidor, monitoreo remoto de procesos, juegos interactivos, chats con retroalimentación del estado de los usuarios, entre otras aplicaciones, como se aprecia en la tabla 22.

Tabla 22.
Casos de aplicación WebSocket

Áreas	Ámbito de aplicación
HOGARES INTELIGENTES	WebSocket es usado en integración con otros protocolos como CoAP y MQTT para proveer soluciones Smarthome: • Administración de luces. Empresa ihome y EVRYTHNG proveen una solución para la integración del protocolo MQTT y la web a través del protocolo Websocket (EVRYTHNG INC., 2016). • Control de flujo de aire. En (A Prototype Air Flow Control System for Home), presentan un prototipo de sistema de control de flujo de aire para la automatización del hogar utilizando AWS IoT Core y el protocolo MQTT sobre el servidor Websocket. (Ozvural & Kurt, 2015)
AMBIENTES INTELIGENTES	WebSocket es usado con Zigbee en ciudades para automatizar actividades como: • Monitoreo ambiental. Un sistema eficiente de monitoreo ambiental basado en ZigBee-WebSocket M2M, en el cual se utiliza una red de sensores ZigBee, para recopilar la información de temperatura y humedad, que utiliza WebSocket para enviar la información recibida por un servidor web al navegador del cliente. (Shuang, Shan, Sheng, & Zhu, 2014) • Estaciones de servicio. Sistema de servicio de Internet orientado a estaciones de servicio basado en Intel Minnowboard, el cual busca recopilar información de tanques de aceite y pistolas de aceite en una estación de servicio (Cui, Xu, Jiang, & Fang, 2016).

6.4.9 Otros aspectos

6.4.9.1 Seguridad El protocolo WebSocket emplea el modelo de origen que utiliza los navegadores web, para implementar seguridad desde una página web. Esto le permite desde el navegador web restringir a que página se le permite establecer una conexión, evitando así vulnerabilidades de código inyectado (Barth, 2011). Cuando el cliente utiliza directamente el protocolo WebSocket (no desde un navegador web), el modelo de origen no es útil. En la sección 10 de (Fette & Melnikov, 2011), se describen algunas consideraciones de seguridad aplicables al protocolo WebSocket, entre las cuales cabe mencionar:

- ✓ El servidor debe validar todas las entradas para clientes que no son navegadores.
- ✓ Los servidores deben verificar que el campo de encabezado origen |***origin***| corresponde a un origen aceptable o esperado en el handshake inicial antes del establecimiento de la conexión.
- ✓ Cifrado de los datos que van del cliente al servidor, así si estos son interceptados, el atacante no pueda acceder a la información de estos para construir una solicitud HTTP falsa.
- ✓ Se recomienda implementar mecanismos de autenticación para los clientes antes de que se establezca la conexión WebSocket como cookies, autenticación HTTP o TLS.
- ✓ Para la confidencialidad e integridad de la conexión se utiliza WebSocket sobre SSL/TLS (URI "*wss://*").
- ✓ Manejo de datos inválidos. Si un cliente o un servidor reciben un dato que no reconocen o viola alguno de los criterios establecidos, pueden desconectar la conexión.

- ✓ Uso de SHA-1 en el establecimiento de la conexión WebSocket con el campo de encabezado |*Sec-WebSocket-Accept*|, que es generado y devuelto en el handshake del servidor.

6.5 Protocolo XMPP (Extensible messaging and presence protocol)

XMPP es un protocolo abierto basado en el lenguaje XML, para la comunicación casi en tiempo real, utilizado en la mensajería instantánea, presencia, multichat, llamadas de voz y video, colaboración, middleware ligero, sindicación de contenidos, y enrutamiento XML genérico; estandarizado por la IETF en la RFC 6120 (P. Saint-Andre, 2011a).

Según (Karagiannis et al., 2015), (XMPP Standards Foundation, s/f-a) las principales características del estándar son:

- El protocolo XMPP es gratuito, abierto, público y de fácil comprensión además de estar estandarizado.
- Arquitectura publicación/suscripción (asíncrono) y también arquitectura solicitud/respuesta (síncrono).
- Se ejecuta sobre el protocolo de transporte TCP.
- Puerto 5222 para cliente y 5269 para servidor
- Para la seguridad utiliza TLS para cifrado y SASL para autenticación.
- No proporciona opciones de calidad de servicio QoS.
- Es extensible porque se puede crear una funcionalidad personalizada sobre el núcleo del protocolo, esto debido al lenguaje XML. Es descentralizado al permitir que cualquiera pueda montar su propio servidor XMPP y tomar el control total de sus comunicaciones.

- Es flexible al incluir diferentes usos como: administración de red, sindicación de contenido, herramientas de colaboración, intercambio de archivos, juegos, monitoreo de sistemas remotos, servicios web, middleware liviano, computación en la nube entre otros.
- Es diverso porque varias compañías y proyectos lo utilizan para crear y desplegar aplicaciones en tiempo real y servicios.

6.5.1 Arquitectura

El protocolo XMPP usa la arquitectura cliente servidor descentralizada, en la cual, la comunicación entre clientes se realiza sin un servidor central que gestione el servicio, sino que cualquier cliente puede implementar su propio servidor y unirse a la red.

Es decir, si un cliente desea enviar un mensajea XMPP a otro cliente, inicialmente se comunica con su servidor XMPP y luego esté servidor se conectará directamente con el servidor del otro cliente para finalmente entregar el mensaje a su destinatario (Ayala Bonilla, 2019) como se aprecia en la figura 44. Cabe resaltar que los clientes XMPP solo se comunican con el servidor XMPP de su dominio, que es el encargado de proveer el servicio.

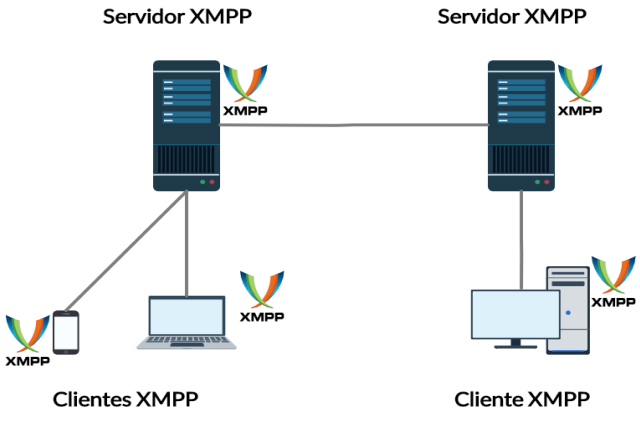


Figura 44. Arquitectura XMPP

Adaptación figura 2.6 (Bosquiazzo, 2016)

6.5.2 Direccionamiento

Cada usuario en la red XMPP requiere un identificador único (Jabber ID o JID), compuesto por un nombre de usuario, un dominio y un recurso de la siguiente forma: ***usuario@dominio/recurso***, donde el usuario es un identificador único dentro de un dominio, el dominio es el servidor donde reside el usuario y el recurso es el cliente desde el que se conecta el usuario (laptop, mobile, home, etc.) (Bas Junquera, 2014).

6.5.3 Comunicación

La comunicación en XMPP se realiza a través de flujos XML o XML streams denotada por la etiqueta <stream> de inicio y la etiqueta </stream> de fin. Un XML stream es un contenedor para el intercambio de elementos XML entre dos entidades en una red y representa el inicio y fin de una comunicación XMPP. El stream se usan para enviar stanzas y puede enviar un número ilimitado de XML stanzas (Bosquiazzo, 2016).

El proceso de comunicación se observa en la figura 45 y se desarrolla de la siguiente manera: Cuando un cliente inicia una sesión con otro cliente o un servidor, se establece una conexión TCP persistente, luego se negocia el XML stream a intercambiar junto con el cifrado y la autenticación, para ello, el destinatario abre un XML stream hacia el receptor y esté responde enviando un XML stream hacia el destinatario, después de establecida la conexión se pueden intercambiar de manera asíncrona y en ambos sentidos paquetes XMPP o stanzas. Finalmente, en cualquier momento tanto el servidor como el cliente pueden cerrar la conexión y terminar la sesión XMPP (Ayala Bonilla, 2019)

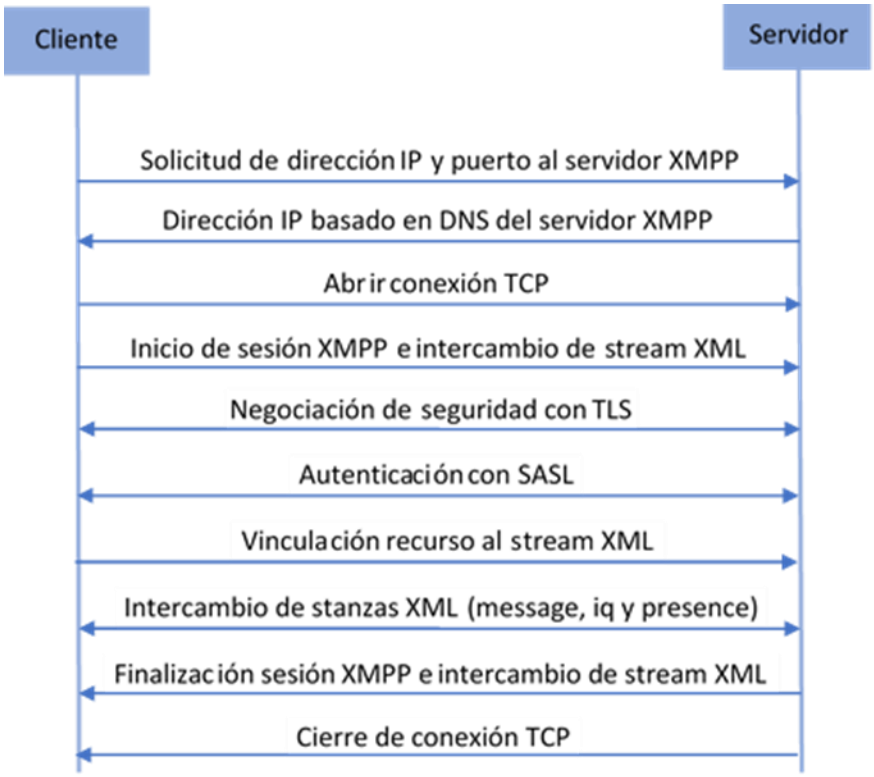


Figura 45. Comunicación XMPP

Adaptación figura 13 (Latvakoski & Heikkinen, 2019), (P. Saint-Andre, 2011a)

La XML stanza es un fragmento de XML con semántica determinada que viaja dentro de un stream XML. Se definen tres tipos de XML stanzas: <message>, <iq> y <presence>, cada una de ellas con cinco atributos comunes: *to*, *from*, id, *type* y *xml:lang*.

- **Message** (<message/>): Es un mecanismo para enviar información de una entidad a otra, generalmente en forma de texto, no requiere ACK y se usan básicamente para la mensajería instantánea y notificaciones. Existen los siguientes tipos: Normal, Chat, GroupChat, HeadLine y Error (Bas Junquera, 2014).

Un mensaje básico como el de la figura 46 consta de los siguientes atributos: *from, to, id y xml:Lang*; que indican el remitente, destinatario, la identificación del usuario y el idioma

respectivamente; además puede contener otros elementos como <*body/*> para especificar el contenido del mensaje o <subject/>, que indica el asunto del mensaje (Bosquiazzo, 2016).

```
<message
        from=' juliet@example.com/balcon '
        id= ' ju2ba41c '
        to= ' romeo@example.net '
        type= 'chat '
        xml : lang= ' es '>
  <body>¿No eres Romeo y un Montague?</body>
</message>
```

Figura 46. Mensaje básico XMPP

Adaptado de la sección 9.1.4 de (P. Saint-Andre, 2011a)

- **Presence** (<presence/>): Es un mecanismo para informar de la disponibilidad de comunicación de un usuario a sus contactos. Para que un usuario pueda ver el estado de otro usuario, se debe suscribir a él y recibir autorización (Bas Junquera, 2014).

La stanza presencia posee el atributo *from* que indica el emisor de la presencia y *type* que indica el estado de la conexión dependiendo de los valores available (conectado) o unavailable (desconectado); además puede poseer un elemento *show* que indica la disponibilidad del usuario, con estados como: chat (disponible para chatear), away (el usuario esta fuera por un corto periodo de tiempo), xa (extended away, fuera por un periodo prolongado de tiempo) y dnd (do not distrub, el usuario no desea recibir mensajes); el elemento *status* con texto libre para mostrar al resto de los contactos. En la figura 47 se presenta un ejemplo de la stanza presencia.

```
<presence
        from='juliet@example.com/balcon'
        type='available'>
         <show>xa</show>
         <status> en la madriguera del conejo! </status>
</presence>
```

Figura 47. Mensaje de presencia XMPP
Adaptación figura 2.9 (Bosquiazzo, 2016)

- **Iq** (<iq/>): Es un mecanismo de solicitud/respuesta que permite a una entidad hacer una solicitud de información a otra entidad. Existen 4 tipos: Get, Set, Result y Error. Esta stanza se usa para conseguir rosters (manejo de contactos), suscripciones a usuarios o a salas de chat, etc (Bas Junquera, 2014).

En la figura 48 se presenta un ejemplo de la stanza iq utilizada para obtener una lista de contactos de un usuario, con el uso de la extensión jabber:iq:roster. El atributo type con el valor get indica que se está solicitando información, el atributo from indica el emisor de la solicitud y el atributo *id* identifica la petición dentro de todo el XML stream.

```
<iq
  from=' juliet@capulet.com/balcon '
  id=' h83vxa4c '
  type='get'>
  <query xmlns = 'jabber:iq:roster'/>
</iq>
```

Figura 48. Mensaje iq XMPP
Adaptado de la sección 8.4 de (P. Saint-Andre, 2011a)

6.5.4 Características extendidas

El IETF ha desarrollado las siguientes especificaciones básicas para XMPP:

- **RFC 5122 - Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP).** Define el uso de identificadores de recursos internacionalizados (IRI) e identificadores uniformes de

recursos (URI) para identificar o interactuar con entidades que pueden comunicarse a través del protocolo de mensajería y presencia extensible (XMPP). (P. Saint-Andre, 2008)

- **RFC 6121 - Extensible messaging and presence protocol (XMPP): instant messaging and presence.** Define las extensiones de las características principales del Protocolo de presencia y mensajería extensible (XMPP) que proporcionan mensajería instantánea básica (IM) y funcionalidad de presencia de conformidad con los requisitos de RFC 2779. (P. Saint-Andre, 2011b)
- **RFC 7590 Use of transport layer security (TLS) in the extensible messaging and presence protocol (XMPP).** Define el uso del protocolo SSL/TLS, respaldo a versiones inferiores, compresión de capa TLS, reanudación de sesión TLS, conjuntos de cifrado, longitudes de clave pública, secreto directo y otros aspectos del uso de TLS con XMPP. Actualiza el RFC 6120. (P. Saint-Andre, 2015b)
- **RFC 7622 - xtensible messaging and presence protocol (XMPP): address format.** Define el formato de dirección para el Protocolo extensible de mensajería y presencia (XMPP), incluido el soporte para puntos de código fuera del rango ASCII. Actualiza el RFC 6122. (P. Saint-Andre, 2015a)

XMPP Standards Foundation (XSF) es la fundación que se encarga de desarrollar extensiones para el protocolo XMPP, las cuales nombra con el prefijo XEP que significa XMPP Extension Protocol. En (XMPP Standards Foundation, s/f-b), se enumeran 435 extensiones para el protocolo XMPP, las cuales se encuentran en diferentes estados como son activo, borrador y final, entre otros. En la tabla 23, se definen algunas extensiones:

Tabla 23.
Extensiones del Protocolo XMPP

Extensión	Descripción
XEP-0001	Define formalmente la naturaleza de los Protocolos de extensión XMPP y los mecanismos para administrarlos y avanzarlos dentro del proceso de estándares de XSF. Estado Activa de tipo procesal. (P. Saint-Andre, Cridland, & Meijer, 2019)
XEP-0402 PEP Native Bookmarks	Esta especificación define una sintaxis y un perfil de almacenamiento para mantener una lista de marcadores de sala de chat en el servidor. Estado Borrador. (Cridland & Brand, 2020)
XEP-0345 Form of Membership Applications	Esta especificación describe la forma y el contenido obligatorio de las solicitudes de membresía. Estado Activa. (Cridland, 2020)
XEP-0199 XMPP Ping	Esta especificación define una extensión de protocolo XMPP para enviar pings de nivel de aplicación a través de secuencias XML. Estado Final. (Peter Saint-Andre, 2019)

6.5.5 Implementaciones

En la tabla 24 se presenta información sobre algunas implementaciones XMPP que han sido desarrolladas en diferentes lenguajes.

Tabla 24.
Implementaciones del protocolo XMPP

Implementación	Descripción
Adium	Es una aplicación cliente de mensajería instantánea gratuita y de código abierto para Mac OS X que se puede conectar a AIM, XMPP (Jabber), ICQ, IRC y más. Escrita utilizando la API Cocoa de Mac OS X. Licencia GNU GPL. Desarrollada por el equipo Adium. (Adium, s/f)
AstraChat	AstraChat es una solución Cliente/Servidor confiable de comunicación y chat multiplataforma móvil que le permite intercambiar información y datos de forma segura en tiempo real. Implementa el RFC 3920, RFC 3921, XEP-0012, XEP-0016, entre otros. Es ofrecido por Rockliffe. (Rockliffe Systems, 2019)
Jackal	Es un servidor XMPP gratuito, de código abierto y de alto rendimiento. Escrito en Go. Permite personalizar, protocolo SSL/TLS forzado, compresión de flujo (zlib), conectividad de base de datos (BadgerDB, MySQL 5.7+, MariaDB 10.2+, PostgreSQL 9.5+), multiplataforma (OS X, Linux). Licencia GPL. (Whited, 2020)

Isode M-Link	Es un servidor basado en el estándar XMPP. Maneja compatibilidad con la confidencialidad de datos mediante TLS y compatibilidad con autenticación SASL. Implementa XEP-0124, admite Publicar/Suscribir. No es compatible con redes restringidas. Ofrecido por Isode. (Isode, 2020)
Sharp.Xmpp	Es un repositorio desarrollado en .NET, fácil de usar. Ofrece soporte para la mayoría de las extensiones del protocolo. Licencia MIT. (pgstath, 2016)
Artalk Xmpp	Es una biblioteca que implementa completamente las especificaciones XMPP Core y XMPP IM. Escrita en C#. Tiene soporte TLS/SSL, autenticación SASL, SOCKS5 y transferencia de archivos en banda, entre otras. Licencia MIT. (Heidari, 2019)
Aioxmpp	Es una biblioteca XMPP, escrita en python puro. Utiliza el módulo de biblioteca estándar asyncio de Python 3.4. compatible con el RFC 6121, utiliza TLS de forma predeterminada. Basada en XEP, soporta XEP-0198, XEP-0060 (PUB/SUB), XEP-0050, XEP-0115, XEP-0045, entre otras. (Schäfer, 2020)

6.5.6 Casos de aplicación

El protocolo XMPP es utilizado en varias áreas de aplicación; en la tabla 25 se aprecian algunos casos de uso que cumplen las necesidades de automatización en determinados ámbitos.

Tabla 25.

Casos de aplicación XMPP

Área	Ámbito de aplicación
EDIFICIOS INTELIGENTES	XMPP es usado en edificios para automatizar actividades como: • Monitoreo del Control de acceso y seguridad. (Rosales Nuñez, 2015)
AMBIENTE INTELIGENTE	XMPP es usado en el medio ambiente para automatizar actividades como: • Monitorización y medición de la calidad del aire en aspectos como la temperatura, el esmog producido por los autos y sustancias como benceno, amoníaco, dióxido de azufre en el medio ambiente. (Sarjerao & Prakasarao, 2018)

SALUD	XMPP es usado en la salud para automatizar actividades como: • Monitoreo remoto continuo y notificación instantánea de signos vitales en pacientes, como la temperatura, frecuencia cardiaca, EGG, etc. (Pawara, 2017)
CIUDAD INTELIGENTE	XMPP es usado en la ciudad para automatizar actividades como: • Monitoreo y seguimiento a sistemas de transporte público en tiempo real por medio de aplicaciones móviles. (Farkas, Nagy, Tomas, & Szabo, 2014) • Seguimiento y control de las necesidades de un automóvil para su correcto funcionamiento y posibilidad de suscribirse a información con entes externos como las estaciones de servicio cercanas, a través de una aplicación móvil. (Bendel et al., 2013) • Estacionamiento Inteligente de vehículos. (Kayal & Perros, 2017) • Realizar monitoreo de seguridad por medio de video vigilancia inteligente. (Sultana & Wahid, 2019)

6.5.7 Otros aspectos

6.5.7.1 Seguridad En una comunicación con XMPP se puede garantizar la integridad y confidencialidad de los mensajes transmitidos con el uso obligatorio de cifrado TLS junto con el mecanismo de autenticación SASL. A pesar de ello, con TLS solo se garantiza cifrado desde de un extremo a otro siempre y cuando no se requiera atravesar por diversos flujos para llegar desde el transmisor hasta el receptor. (P. Saint-Andre, 2011a)

Sin embargo, según (P. Saint-Andre, 2011a) existe la posibilidad de abarcar otros tipos de mecanismos diferentes a los mencionados anteriormente. Lo que ha incentivado a la implementación de nuevas alternativas que propicien la seguridad con el uso del protocolo XMPP, como las presentadas a continuación:

En (Aros & Torres, 2018), los autores implementaron un mecanismo para disminuir los ataques de seguridad en la red, generando permisos que permitan la creación de identidades y el registro en banda de clientes a servidores y de servidores a servidores, por medio de las

extensiones ya existentes XEP-007 y XEP-0348; pero con mejoras en la inclusión de restricciones para evitar que cualquiera pueda acceder a la red. Las pruebas se aplicaron sobre el sistema de mensajería instantánea de código abierto OpenFire, comprobando su eficacia obteniendo como resultado cero ataques en la red.

En (Järvinen et al., 2018), se habla sobre el etiquetado de datos como beneficio para facilitar su procesamiento y búsqueda. En aspectos de seguridad, los autores proponen un mecanismo que genere interoperabilidad entre los diferentes dominios destinados al uso de etiquetas para seguridad; como los basados en XEP-0258 y STANAG 4778. Con el objetivo de reducir la necesidad de crear aplicaciones para cada tipo de seguridad, utilizando un solo etiquetado general.

En (Guo, Wu, Xia, & Li, 2015), los autores profundizan en la seguridad de comunicaciones con XMPP para redes de sensores basadas en la norma IEEE 21451, implementando el método de autenticación una vez contraseña (OTP) para ambos lados, permitiendo crear roles en cada nodo y de esta manera determinar los permisos adecuados para clientes y servidores.

6.6 Protocolo HTTP/2 (Hypertext transfer protocol version 2)

HTTP/2 es un protocolo que permite la transferencia de archivos a través de la web, fue desarrollado por la IETF y presentado en el 2015 a trasvés de la RFC 7540, adopta las características principales de su versión predecesora HTTP/1.1. No obstante, tiene el objetivo de mejorar su funcionalidad, garantizando una transferencia de forma ágil, además de manejar de manera eficiente los recursos de red y la reducción de latencia. (Londoño, González, Céspedes, Bustos, & Montenegro, 2017)

Las principales características del protocolo según (Mike Belshe, Peon, & Thomson, 2015), (Stenberg, 2014), (Imperva, 2016) y (Fornaroli, Tugnarelli, Santana, & Díaz, 2018) son:

- Puertos predeterminados 80/443 (URLs http/URLs https)
- Basado en la arquitectura Cliente/Servidor.
- Uso obligatorio de TLS.
- Mantiene la misma semántica y compatibilidad que sus versiones previas.
- Los mensajes son enviados por medio de tramas.
- Utiliza multiplexación de streams de las solicitudes y respuestas para reducción de la latencia.
- Para disminuir la sobrecarga utiliza la compresión y codificación binaria HPACK.
- Maneja prioridades en solicitudes y capacidades.
- Posee sistema de control de flujo, manejo de errores y actualizaciones.
- Envía de forma anticipada recursos al cliente antes de ser solicitados, con la funcionalidad Server Push.

6.6.1 Estructura y funcionamiento

En la figura 49 se presenta la estructura de la capa binaria del protocolo HTTP/2, en la cual se observa que mantiene la semántica (verbos, métodos, códigos de estado, URI, encabezados, etc.) utilizada en el protocolo HTTP/1.1. Difieren en la forma en cómo se realiza el encapsulamiento a través de tramas pequeñas en formato binario; como se aprecia en la figura 49, en donde las tramas HEADER y DATA son las utilizadas para enviar encabezados y datos respectivamente. (Grigorik, 2013)

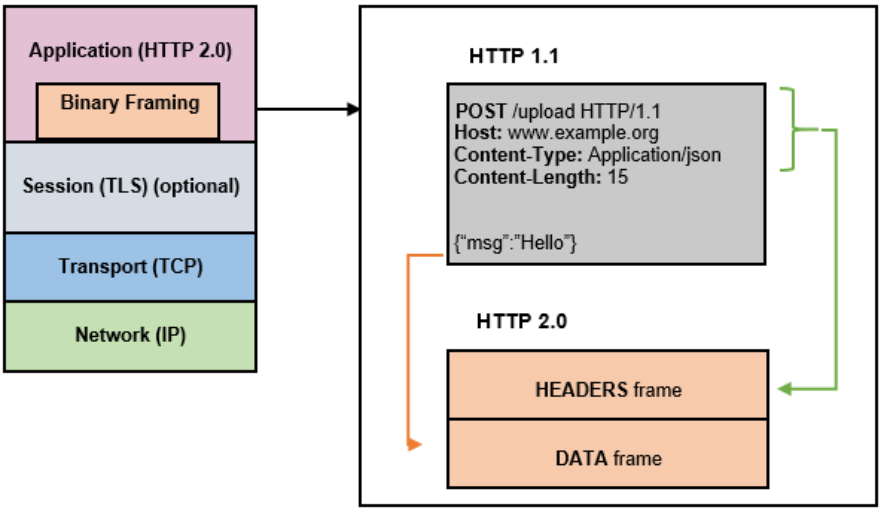


Figura 49. Capa de Trama Binaria HTTP/2
Adaptado de la figura 12-1 de (Grigorik, 2013)

6.6.2 Conexión en HTTP/2 (Hypertext transfer protocol version 2)

En HTTP/2 se realiza una conexión por TCP utilizando los esquemas de HTTP/1.1, en donde las URLS "http:// "y "https:// "se mantienen. Para que una conexión sea exitosa se requiere que tanto el cliente como el servidor utilicen un mecanismo de codificación binaria, por lo tanto, HTTP/2 debe ser actualizado por medio de la cabecera "Upgrade". Según (Marín Pérez, 2018) existen tres formas de adquirir recursos vía HTTP/2 desde un servidor como son:

- Conexión desde http sin conocimiento: El cliente realiza una petición http desconociendo si el servidor soporta HTTP/2, usando la cabecera "Upgrade" y en su contenido el identificador "h2c" lo que significa que no usa TLS, adicional debe incluir una línea de cabecera HTTP2-Settings. Si el servidor no soporta HTTP/2 emite la respuesta "HTTP/1.1 200 OK", pero si el servidor si soporta HTTP/2 emite la respuesta "HTTP/1.1 101". Una vez recibida la respuesta, el cliente debe enviar un prólogo como confirmación del protocolo en uso y establecer una conexión con las opciones determinadas inicialmente.

- Conexión desde https sin conocimiento: El cliente realiza una petición https desconociendo si el servidor soporta HTTP/2, usando la cabecera "Upgrade" y en su contenido el identificador "h2" lo que significa que usa TLS por medio de una extensión ALPN (Negociación de Protocolo de Capa de Aplicación). Al terminar la negociación, se envían prólogo de confirmación del protocolo en uso, desde las dos partes.
- Conexión http/2 con conocimiento previo: Cuando un cliente conoce que el servidor soporta HTTP /2, envía prólogo de conexión y de la misma manera el servidor como respuesta emite prólogo de conexión e inician el envío de tramas, permitiendo así reducir el tiempo de comunicación, aumentando su capacidad. El cliente puede descubrirlo con anticipación por medio del encabezado de respuesta ALT-SVC que anuncia los servicios alternativos a los cuales se pueden acceder.

6.6.3 Estructura y tipos de trama

En HTTP/2 se utilizan 10 tipos de tramas que manejan parámetros específicos de acuerdo con su funcionalidad. Sin embargo, tienen en común la estructura definida en la figura 50.

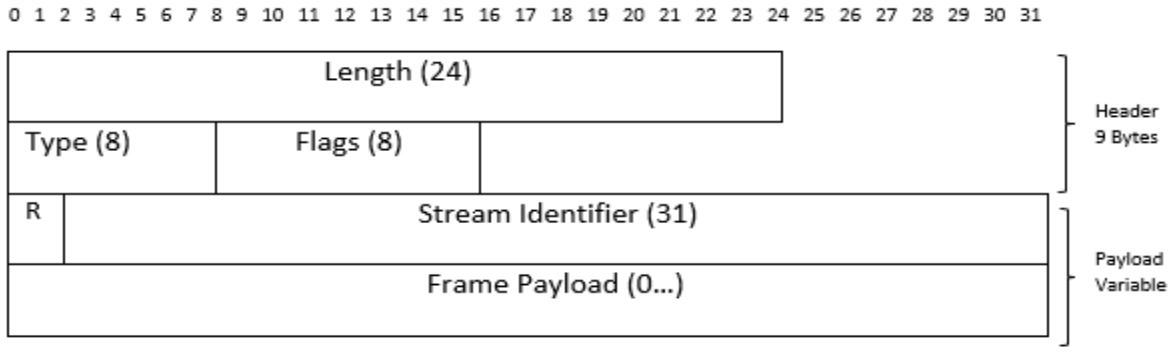


Figura 50. Trama HTTP/2
Adaptado de la figura 6 de (Marín Pérez, 2018)

Los campos de una trama HTTP/2 según (Marín Pérez, 2018) son:

- Length: Tiene un tamaño de 25 bytes sin signo, especifica la longitud del Payload sin incluir los primeros 9 bytes de la cabecera de la trama.
- Type: Tiene un tamaño de 8 bits y especifica el tipo de trama.
- Flags: Tiene un tamaño de 8 bits reservados para funciones booleanas específicas de cada tipo de trama.
- R: Campo de un bit reservado.
- Stream Identifier: Tiene un tamaño de 31 bits sin signo, en el cual se identifica el flujo al que pertenece la trama.
- Frame Payload: Tiene un tamaño variable, el cual depende del tipo de trama; determina el contenido de la trama y su longitud está definida por el primer campo "Length"

A continuación, en la tabla 26 se presenta una breve descripción de cada una de las 10 tramas HTTP/2. Para más información en cuanto a parámetros y funcionalidad consultar la sección 6 de la RFC7540 de 2015.

Tabla 26.

Tipos de Tramas HTTP/2

Tipos de trama	Descripción	Campo Type
DATA	Utilizada para transmitir los recursos de la capa de aplicación entre un cliente y un servidor.	0x0
HEADERS	Utilizada para iniciar un stream, en el intercambio de peticiones y respuestas. En su Payload se incluyen los encabezados HTTP/2.	0x1
PRIORITY	Utilizada para indicar la prioridad de un stream, además de poder cambiar prioridades anteriores.	0x3
RST STREAM	Utilizada para finalizar de forma inmediata un stream, informando a través de un código de error al destinatario.	0x3

SETTINGS	Utilizada para realizar la configuración de una comunicación entre dos extremos.	0x4
PUSH PROMISE	Utilizada para notificar que se va a enviar un stream justo antes de enviarlo.	0x5
PING	Utilizada para medir el tiempo mínimo entre los extremos y para saber si un extremo es funcional.	0x6
GOAWAY	Utilizada para detectar nuevos flujos y cerrar una conexión.	0x7
WINDOW UPDATE	Utilizada para realizar el control del flujo por conexión y stream.	0x8
CONTINUATION	Utilizada para enviar las cabeceras que no se han podido transmitir anteriormente.	0x9

Nota: Adaptación a partir de (Marín Pérez, 2018)

6.6.4. Multiplexación de stream

Un Stream hace referencia a un flujo bidireccional de tramas independientes transmitidas entre un cliente y un servidor en una única conexión TCP (Ruz Mieres, 2019), tanto un cliente como un servidor pueden crear un flujo en una comunicación y de igual manera cerrarlo en cualquier momento.

La capa de trama binaria de HTTP/2 permite la Multiplexación de Stream otorgando la opción a los clientes y servidores de dividir un mensaje HTTP en tramas independientes con la posibilidad de intercalarlos y volverlos a unir en el otro extremo, como se aprecia en la Figura 51; en donde por medio de una única conexión, mientras el servidor responde al cliente solicitudes en diferentes tramas asociadas a los Stream1 y Stream2, el cliente está enviando una solicitud a través del Stream5 al servidor, ejecutándose tres flujos en paralelo al mismo tiempo. (Grigorik, 2013)

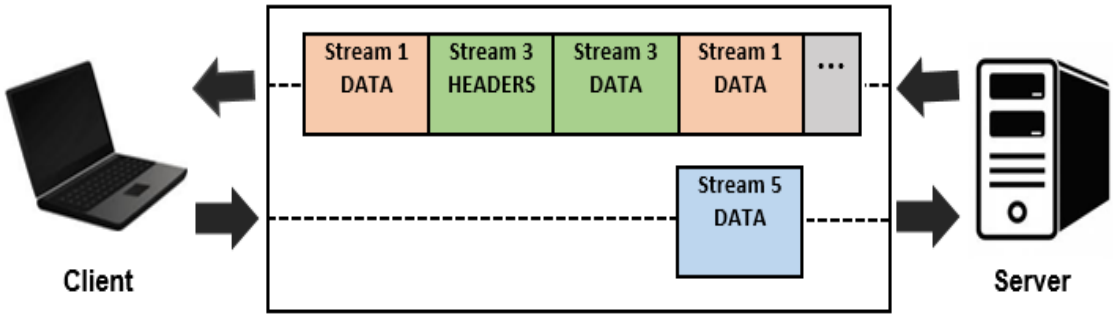


Figura 51. Multiplexación en una conexión HTTP/2
Adaptado de la figura 2.4 de (Ruz Mieres, 2019)

La propiedad de Multiplexación de Stream en una conexión HTTP/2 ofrece varios beneficios según (Grigorik, 2013), entre los que se encuentran:

- Permite intercalar múltiples solicitudes y respuestas en paralelo sin generar bloqueos.
- A diferencia de HTTP/1.X que requiere el uso de múltiples conexiones TCP para cada solicitud y respuesta, en HTTP/2 la comunicación se realiza en una sola conexión TCP.
- Elimina la latencia innecesaria mejorando la capacidad de red y eficientes tiempos de carga de páginas web.

Además, para la administración y control de flujo, tanto el servidor como el cliente pueden enviar una trama WINDOWS_UPDATE al receptor para establecer la cantidad de bytes que desea recibir, esto con el objetivo de distribuir el ancho de banda entre los flujos y evitar que el ancho de banda se agote. (Marín Pérez, 2018).

6.6.5 Priorización de flujo

Un cliente en HTTP/2 tiene la posibilidad de estimar cuales flujos son más importantes que otros durante una conexión, estableciendo la dependencia entre flujos y la asignación de su peso entre 1 y 152. Lo anterior, le permite la construcción de un árbol de priorización para dar a conocer al servidor la manera en cómo desea recibir las respuestas y al mismo tiempo faculta al servidor

para controlar la asignación de recursos como memoria, ancho de banda, entre otros, de manera que asegure la entrega de respuestas de prioridad. (Grigorik, 2013)

6.6.6 Compresión de cabeceras

Durante una comunicación en HTTP/1.1 cada mensaje transmitido contiene un encabezado con los metadatos que describen sus características, debido a su formato en texto plano sin ningún tipo de codificación, ocasiona un consumo amplio de ancho de banda generando sobrecargas. A diferencia de HTTP/2 que maneja formato binario y utiliza el método de compresión de cabeceras HPACK, el cual permite codificar los encabezados usando el código Huffman estático. (Grigorik, 2013)

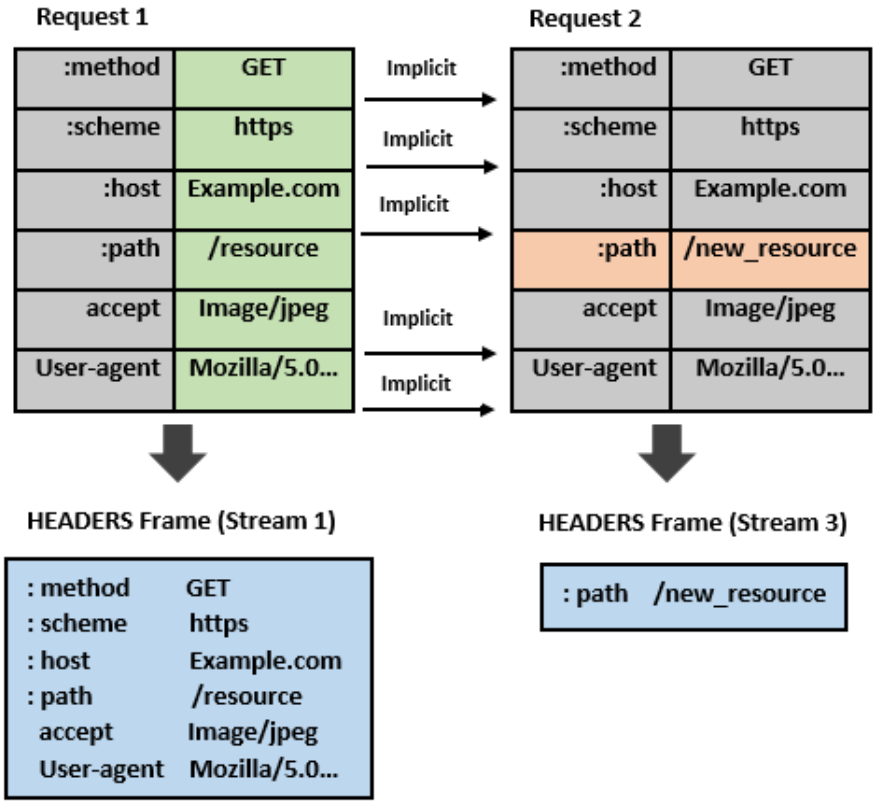


Figura 52. Compresión de cabeceras para HTTP/2
Adaptado de la figura 12-16 de (Grigorik, 2013)

En la codificación Huffman se requiere que el cliente y el servidor conserven una lista de campos comunes de encabezados HTTP que componen una tabla llamada estática, interviene también una tabla dinámica que inicialmente está vacía y se actualiza con los campos que intercambia durante una conexión como se precia en la figura 52. En donde cinco campos de la solicitud 1 que ya han sido transferidos anteriormente, actualizan campos de la solicitud 2, permitiendo reducir su tamaño enviando únicamente por el Stream 3 el campo "path: /new_resource". (Grigorik, 2013)

6.6.7 Server push

Server push es un mecanismo de HTTP/2 que permite al servidor enviar múltiples respuestas a una solicitud de un cliente antes de ser requeridas, como se aprecia en la figura 53. (Grigorik, 2013). Lo que quiere decir que se envían recursos adicionales a la solicitud inicial con el objetivo de optimizar el rendimiento en el cargue de las páginas web. No obstante, se requiere de un mayor consumo de ancho de banda lo que podría ocasionar mayor consumo de batería en aplicaciones IoT. (Ruz Mieres, 2019)

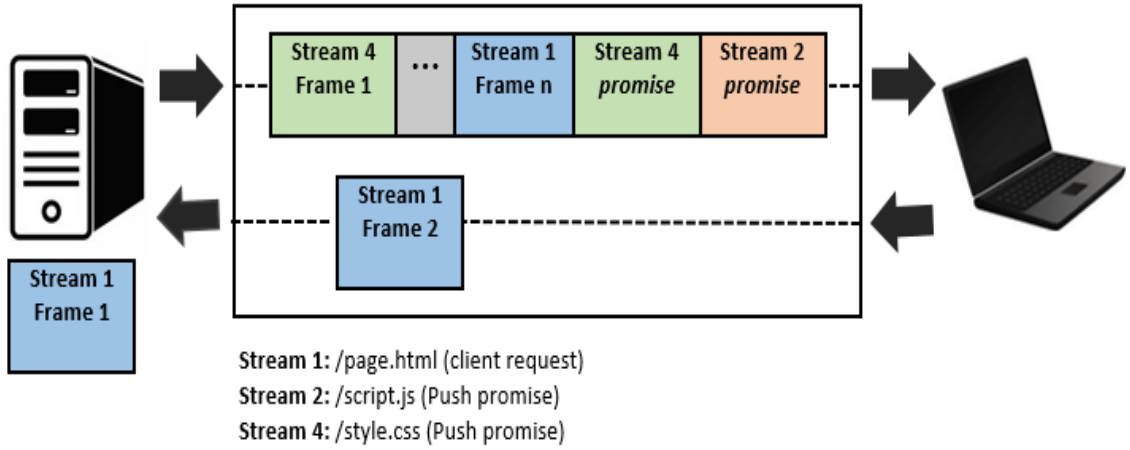


Figura 53. Funcionamiento de Server Push en HTTP/2
Adaptado de la figura 12-5 de (Grigorik, 2013)

6.6.8 Características extendidas

Existen características adicionales que mejoran las capacidades del protocolo HTTP/2 entre las que se encuentran:

- **RFC 7541 HPACK (Header compression for HTTP/2):** Esta especificación define un formato de compresión que elimina campos de encabezado redundantes, limita la vulnerabilidad a ataques de seguridad conocidos y tiene un requisito de memoria limitada para su uso en entornos restringidos. (Peon & Ruellan, 2015)
- **RFC 8336 (The origin HTTP/2 frame)** Este documento especifica la trama ORIGIN para HTTP/2, para indicar qué orígenes están disponibles en una conexión determinada. (Nottingham & Nygren, 2018)
- **RFC 8441 Bootstrapping WebSockets with HTTP/2:** Este documento define un mecanismo para ejecutar el protocolo WebSocket (RFC 6455) en una sola secuencia de una conexión HTTP/2. (McManus, 2018)
- **RFC 8740 Using TLS 1.3 with HTTP/2:** Este documento actualiza la RFC 7540 al prohibir la autenticación posterior al protocolo de enlace TLS 1.3, como un análogo a la restricción de renegociación TLS 1.2 existente. (Benjamin, 2020)

6.6.9 Implementaciones

En la actualidad varios navegadores admiten el uso de HTTP/2 según (Deveria, 2020) estos son: Edge, Firefox, Chrome, Safari, Opera, entre otros. En la tabla 27 se presentan algunas de las implementaciones para HTTP/2.

Tabla 27.

Implementaciones HTTP/2

Nghttp2	Es una implementación de HTTP/2 para cliente, servidor y proxy, escrita en C, con licencia MIT, utiliza la compresión de encabezado para HTTP/2 HPACK (RFC 7541), posee un codificador y decodificador HPACK disponibles como API pública. Permite negociar la conexión con TLS, ALPN, NPN, actualizar la conexión o iniciar directamente una conexión (Tsujikawa, 2020).
http2	Es una biblioteca escrita en Go para cliente y servidor, con licencia MIT, permite negociar la conexión con ALPN o actualizarla, control de flujo, multiplexación de solicitudes y apagado seguro (paku, 2016).
H20	Es un servidor HTPP escrito en C con licencia MIT, con soporte para HTTP/1.x, HTTP/2 y HTTP/3 (experimental), con soporte completo para la dependencia y la priorización, servidor push, admite TLS, publicación de archivos estáticos, proxy inverso, entre otras (DeNA Co. Ltd., 2019).
HTTP-2	Es una implementación escrita en Rubi con licencia MIT, para cliente y servidor, con compresión de encabezado HPACK, soporte para entramado binario y codificación, con multiplexación de múltiples solicitudes y respuestas, priorización de flujo, servidor push, control de flujo y conexión (Grigorik, 2019).
Deuterium	Es una implementación ligera de HTTP/2 ideal para IoT, escrito en C, para servidor y cliente, admite TLS, negociación directa y con ALPN, altamente personalizable incluye número de conexiones y numero de transmisiones por conexión (Simpson, 2019).
Jetty	Es un servidor web ligero escrito en Java y un contenedor de servlets basado en estándares, desarrollado por la Fundación Eclipse con licencia Apache 2.0, para cliente y servidor, es asíncrono y escalable, admite TLS/SSL (Lamy, 2020) (Eclipse Foundation, 2016).
Armeria	Es una biblioteca cliente servidor asíncrona, construida sobre Java 8, y las librerías Netty, Thrift y gRPC, con licencia Apache 2.0, Admite HTTP/2 tanto en TLS como en conexiones de texto sin cifrar, admite actualización de HTTP/2, soporte de proxy, ideal para construir microservicios asíncronos (LINE Corporation, 2020).

6.6.10 Casos de aplicación

El protocolo HTTP/2 presenta varias características (Montenegro, Cespedes, Loreto, & Simpson, 2016) que le podrían permitir operar en aplicaciones IoT (J.David de Hoz et al., 2018), (De Hoz Diego, Saldana, Fernandez-Navajas, & Ruiz-Mas, 2019). A pesar de eso, no se evidenciaron casos de usos o aplicaciones actualmente del protocolo HTTP/2 para IoT.

6.6.11 Otros aspectos

6.6.11.1 Seguridad El protocolo HTTP/2 al ser de formato binario elimina los problemas de seguridad que vienen con los protocolos basados en texto. Es soportado por casi todos los navegadores web (Deveria, 2020). Según (M. Belshe, Peon, & Thomson, 2015), algunas de las consideraciones de seguridad del protocolo HTTP/2 a tener en cuenta son las siguientes:

- ✓ Debe utilizarse el cifrado en TLS con un identificador ALPN para protegerse de ataques de protocolo cruzado. Las implementaciones de HTTP/2 deben utilizar TLS versión 1.2 o superior.
- ✓ Deben tratarse como malformadas, las solicitudes o respuestas que contengan un nombre de campo de encabezado no valido para evitar los ataques intermedios de encapsulación.
- ✓ No deben almacenarse en cache, respuestas para las que un servidor de origen no tiene autoridad.
- ✓ No debe utilizarse la compresión, si la fuente de datos no se puede determinar de manera confiable.
- ✓ Utiliza el padding para ocultar el tamaño exacto del contenido de la trama y para mitigar ataques dentro de HTTP.

- ✓ El algoritmo HPACK (Peon & Ruellan, 2015) permite que HTTP/2 evite las amenazas de seguridad prevalentes dirigidas a protocolos de capa de aplicación basados en texto.

7. Conclusiones y Recomendaciones

En este trabajo se realizó una revisión literaria de protocolos de capa de aplicación para Internet de las cosas (IoT) en redes IEEE 802.15.4. Lo más importante de esta revisión literaria fue conocer cómo operan estos protocolos bajo los requerimientos que imponen las redes de baja potencia y con pérdidas.

Con la investigación realizada se evidenció que en algunos casos de aplicación sus autores recomendaron a partir de sus experiencias, el uso de los protocolos CoAP y MQTT para redes de baja potencia y con pérdidas. CoAP es empleado por ser ligero debido a que utiliza UDP en su capa de transporte al no utilizar muchos recursos; sin embargo, no proporciona garantía de entrega de mensaje y no evita la duplicación de información. MQTT es liviano, debido a su encabezado pequeño, sin embargo, al usar TCP puede generar sobrecarga y por ende un tamaño extenso de los mensajes; además, es inseguro y de baja interoperabilidad.

De los protocolos orientados a mensajería AMQP y XMPP no son óptimos para redes limitadas. AMQP debido a su soporte para seguridad, confiabilidad, aprovisionamiento e interoperabilidad aumenta la sobrecarga y el tamaño del mensaje; lo que resulta poco adecuado para aplicaciones IoT con dispositivos de bajos recursos. Así mismo, XMPP está diseñado para dispositivos ricos en recursos; sin embargo, posee la extensión XMPP-IoT para el caso de aplicaciones restringidas.

Websocket es un protocolo para transferencia web ligero que reduce la sobrecarga optimizando el uso de la red. Aunque no está diseñado para dispositivos restringidos puede ser una solución para aplicaciones IoT mediante la integración con otros protocolos como CoAP,

MQTT y AMQP, proporcionando transporte de mensajería en navegadores web y seguridad. A pesar de eso no se evidenció el uso del protocolo en redes LLN.

HTTP/2 es un protocolo utilizado para la transferencia de archivos en entornos web; diseñado para mejorar las funcionalidades de su antecesor HTTP 1.1, permitiendo el uso eficiente de los recursos de la red, al permitir múltiples solicitudes y respuestas en una única conexión. Sin embargo, no se evidenciaron casos de aplicación en ningún tipo de red para IoT.

De acuerdo al estudio realizado de los diferentes protocolos de la capa de aplicación de IoT, puede concluirse que las redes IEEE 802.15.4 están pensadas para propósitos muy específicos donde los dispositivos deben operar en unas condiciones extremas en las cuales no pueden operar otras redes, con muy bajas potencias y pérdidas pero que resulta apropiado para crear redes de sensores IoT de bajo costo, por lo tanto, en la capa de aplicación solo aplican protocolos diseñados para estos casos como CoAP y MQTT.

Existen intentos de adaptaciones de estos protocolos, pero consideramos que ese tipo de adaptaciones que se encuentran en estudio posiblemente no brinden grandes ventajas.

Referencias bibliográficas

Adiono, T., Manangkalangi, B. A., Muttaqin, R., Harimurti, S., & Adijarto, W. (2017). Intelligent and secured software application for IoT based smart home. *IEEE 6th Global Conference on Consumer Electronics, GCCE 2017*, *2017-Janua*(Gcce), 1–2. https://doi.org/10.1109/GCCE.2017.8229409

Adiono, T., Marthensa, R., Muttaqin, R., Fuada, S., Harimurti, S., & Adijarto, W. (2017). Design of database and secure communication protocols for Internet of things based smart home system. *IEEE Region 10 Annual International Conference/TENCON*, *2017-Decem*, 1273–1278. https://doi.org/10.1109/TENCON.2017.8228053

Adium. (s/f). Adium - Download. Recuperado el 23 de mayo de 2020, de https://adium.im/

Aires, M. (2017). Socket.io with Elephant.io. Recuperado el 6 de marzo de 2020, de https://github.com/mairesweb/socket.io-elephant.io

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, *17*(4), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

Albano, M., Lino Ferreira, L., Pinho, L. M., & Alkhawaja Rahman, A. (2015). Message-oriented middleware for smart grids. *Computer Standards & Interfaces*, *38*, 133–143. https://doi.org/10.1016/j.csi.2014.08.002

Almazroi, A. A. (2019). Security mechanism in the internet of things by interacting HTTP and MQTT protocols. *IEEE 11th International Conference on Communication Software and Networks, ICCSN 2019*, 181–186. https://doi.org/10.1109/ICCSN.2019.8905262

Amin, Y. M., & Abdel-Hamid, A. T. (2016). Classification and analysis of IEEE 802.15.4 MAC layer attacks. *International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT), Cairo*, 1–8. https://doi.org/10.1109/MoWNet.2016.7496624

Ammar, M., Russello, G., & Crispo, B. (2018). Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications*, *38*, 8–27. https://doi.org/10.1016/j.jisa.2017.11.002

Amsüss, C. (2014). The Python CoAP library. Recuperado el 20 de enero de 2020, de https://github.com/chrysn/aiocoap

Apache ActiveMQ. (2019). ActiveMQ. Recuperado el 2 de marzo de 2020, de https://activemq.apache.org/

Apache Qpid. (2020). Qpid Broker-J. Recuperado el 2 de marzo de 2020, de https://github.com/apache/qpid-broker-j

Apache Qpid™. (2015a). Broker-J. Recuperado el 2 de marzo de 2020, de https://qpid.apache.org/components/broker-j/index.html

Apache Qpid™. (2015b). Qpid C++ Broker. Recuperado el 2 de marzo de 2020, de https://qpid.apache.org/components/cpp-broker/index.html

Apache Qpid™. (2015c). Qpid JMS. Recuperado el 2 de marzo de 2020, de https://qpid.apache.org/components/jms/index.html

Apache Qpid™. (2015d). Qpid Proton. Recuperado el 2 de marzo de 2020, de https://qpid.apache.org/proton/index.html

Arco Castillo, Á., & Navarro Ortiz, J. (2019). *Diseño e implementación de un sistema mqtt sin bróker basado en SDN (Tesis de pregrado).* (Universidad de Granada). Recuperado de https://wpd.ugr.es/~jorgenavarro/thesis/2019_TFG_AlvaroArcoCastillo.pdf

Arequipa Cunalata, D. A. (2019). *Desarrollo de un prototipo de sistema de seguridad contra intrusos utilizando protocolos de IoT sobre la plataroma Zolertia Remote (Tesis de pregrado)*. (Escuela Politécnica Nacional). Recuperado de http://bibdigital.epn.edu.ec/handle/15000/20318

Aros, M., & Torres, R. (2018). Implementing a signing forms mechanism in an open XMPP Server to reduce successful network attacks. *Proceedings of the IV School of Systems and Networks*, *2178*, 79–82. Recuperado de http://ceur-ws.org/Vol-2178/SSN2018_paper_26.pdf

Astorqui, F. (2016). Sensor IoT para monitorización de consumo de energía en continua. *Trabajo Fin de Grado en Ingeniería de Computadores*.

Atmoko, R. A., & Yang, D. (2019). Online Monitoring & Controlling Industrial Arm Robot Using MQTT Protocol. *International Conference on Robotics, Biomimetics, and Intelligent Computational Systems, Robionetics 2018*, 12–16. https://doi.org/10.1109/ROBIONETICS.2018.8674672

Ayala Bonilla, J. F. (2019). *Desarrollo de un sistema para gestión de roles de pago con envío de notificaciones en tiempo real (Tesis de pregrado)*. (Escuela Politécnica Nacional). Recuperado de http://bibdigital.epn.edu.ec/handle/15000/20383

Bahia, J. G., & Campista, M. E. M. (2017). *Um Mecanismo de Controle de Demanda no Provimento de Serviços de IoT Usando CoAP*. 14. Recuperado de https://www.gta.ufrj.br/ftp/gta/TechReports/BaCa17.pdf

Barth, A. (2011). The Web Origin Concept. Recuperado de RFC 6454 website: http://tools.ietf.org/html/rfc6454

Bas Junquera, M. (2014). *Implementación de un gadget OpenSocial para mensajería*

instantánea (Tesis de pregrado). (Univerisdad Carlos III de Madrid). Recuperado de https://e-archivo.uc3m.es/handle/10016/22425

Belcredi, G., Modernell, P., Sosa, N., Steinfeld, L., & Silveira, F. (2015). An implementation of a home energy management platform for Smart Grid. *IEEE PES Innovative Smart Grid Technologies Latin America (ISGT LATAM), Montevideo, Uruguay.*, 270–274. https://doi.org/10.1109/ISGT-LA.2015.7381166

Bellagente, P., Depari, A., Ferrari, P., Flammini, A., Sisinni, E., & Rinaldi, S. (2018). M3IoT - Message-oriented middleware for M-health Internet of Things: Design and validation. *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 1–6. https://doi.org/10.1109/I2MTC.2018.8409656

Belshe, M., Peon, R., & Thomson, M. (2015). Hypertext Transfer Protocol Version 2 (HTTP/2). *RFC 7540*, 1–96. Recuperado de https://tools.ietf.org/html/rfc7540

Belshe, Mike, Peon, R., & Thomson, M. (2015). Hypertext Transfer Protocol version 2 (HTTP/2). *RFC 7540*, 1–96. Recuperado de https://tools.ietf.org/html/rfc7540

Bendel, S., Springer, T., Schuster, D., Schill, A., Ackermann, R., & Ameling, M. (2013). A service infrastructure for the Internet of Things based on XMPP. *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2013*, (March), 385–388. https://doi.org/10.1109/PerComW.2013.6529522

Benjamin, D. (2020). Using TLS 1.3 with HTTP/2. *RFC 8740*, 1–5. Recuperado de https://tools.ietf.org/html/rfc8740

Bevywise Networks Inc. (2020). MQTT Broker with AI / ML Integration using Python. Recuperado el 5 de marzo de 2020, de https://www.bevywise.com/mqtt-broker/

Bezerra, D., Aschoff, R. R., Szabo, G., & Sadok, D. (2018). An IoT protocol evaluation in a

smart factory environment. *Latin American Robotics Symposium, Braziliian Symposium on Robotics (SBR) and Workshop on Robotics in Education (WRE) 2018*, (ii), 118–123. https://doi.org/10.1109/LARS/SBR/WRE.2018.00030

Bharath, M., Reddy, K. V., & Dey, R. (2018). Implementation of IoT architecture for intruder alert system using MQTT protocol and MEAN stack. *4th International Conference on Computing Communication and Automation, ICCCA 2018*, 1–5. https://doi.org/10.1109/CCAA.2018.8777526

Bindle, L., & Adeoye, D. (s/f). MQTT-C. Recuperado el 5 de marzo de 2020, de https://liambindle.ca/MQTT-C/

Borman, C. (2018). Well-Known URIs for the WebSocket Protocol. *RFC 8307*, 1–3. Recuperado de https://tools.ietf.org/html/rfc8307

Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., & B. Raymor, E. (2018). CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. *RFC 8323*, 1–54. Recuperado de https://tools.ietf.org/html/rfc8323

Bormann, C. (2016). CoAP — Constrained Application Protocol | Implementations. Recuperado el 20 de enero de 2020, de https://coap.technology/impls.html

Bormann, Carsten, & Shelby, Z. (2016). Block-Wise Transfers in the Constrained Application Protocol (CoAP). *RFC 7959*, 1–37. Recuperado de https://tools.ietf.org/pdf/rfc7959.pdf

Bosquiazzo, D. A. (2016). *Servcio alternativo y aumentativo de mensjeria instantanea (Tesis de pregrado)*. (Universidad Tecnológica Nacional). Recuperado de https://ria.utn.edu.ar/xmlui/bitstream/handle/20.500.12272/1389/Informe Final PFC - Dario Bosquiazzo.pdf?sequence=1

Candia, A., & Varela, L. N. (2017). *WSN para servicios públicos metropolitanos (Tesis de*

pregrado). (Univesidad Nacional de la Plata). Recuperado de http://sedici.unlp.edu.ar/bitstream/handle/10915/75748/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Cardozo, C., Camps, I., & Martín, D. (2015). *RSItrust Red de Sensores Inalámbricos para monitoreo de condiciones microclimáticas en cultivos de cítricos (Tesis de pregrado).* (Universidad de la República). Recuperado de https://iie.fing.edu.uy/publicaciones/2015/CCD15

Carrillo, A. M. (2017). *Evaluación del desempeño en la transmisión de señales biomédicas en un ambiente inalámbrico en redes 6LoWPAN (Tesis de maestria).* (Centro de Investigación Científica y de Educación Superior de Ensenada , Baja California). Recuperado de https://cicese.repositorioinstitucional.mx/jspui/bitstream/1007/899/1/TesisVersionFinal.pdf

Castro Heredia, J. (2014). *Uso del protocolo CoAP para la implementación de una aplicación domótica con redes de sensores inalámbricas (Tesis de pregrado).* (Universidad Politécnica de Cartagena). Recuperado de http://repositorio.upct.es/bitstream/handle/10317/4163/pfc5908.pdf?sequence=1

Chaudhary, A., Peddoju, S. K., & Kadarla, K. (2017). Study of Internet-of-Things Messaging Protocols Used for Exchanging Data with External Sources. *IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems, MASS, Orlando, FL, 2017*, 666–671. https://doi.org/10.1109/MASS.2017.85

CoAPSharp – Idaax. (2019). Recuperado el 23 de noviembre de 2019, de http://idaax.com/coapsharp/

Constrained Application Protocol. (2019). Recuperado el 23 de noviembre de 2019, de https://en.wikipedia.org/wiki/Constrained_Application_Protocol

Cridland, D. (2020). XEP-0345: Form of Membership Applications. Recuperado el 23 de mayo de 2020, de XMPP Standards Foundation website: https://xmpp.org/extensions/xep-0345.html

Cridland, D., & Brand, J. (2020). XEP-0402: PEP Native Bookmarks. *XMPP Standards Foundation*.

Cui, H. H., Xu, T., Jiang, X. N., & Fang, L. (2016). Gas Stations Oriented Internet Service System Based on Intel Minnowboard. *9th International Symposium on Computational Intelligence and Design, ISCID 2016*, *1*, 290–293. https://doi.org/10.1109/ISCID.2016.1073

De Hoz Diego, J. D., Saldana, J., Fernandez-Navajas, J., & Ruiz-Mas, J. (2019). IoTsafe, Decoupling Security from Applications for a Safer IoT. *IEEE Access*, *7*, 29942–29962. https://doi.org/10.1109/ACCESS.2019.2900939

de Hoz, Jorge David, Saldana, J., Fernandez-Navajas, J., & Ruiz-Mas, J. (2019). Decoupling security from applications in CoAP-based IoT devices. *IEEE Internet of Things Journal*, *7*(1), 10. https://doi.org/10.1109/jiot.2019.2951306

de Mula, I., Ferrari, G., & Firme, G. (2011). *ContikiWSN Estudio, Análisis y Diseño de Redes de Sensores Inalámbricas con Contiki OS* (Universidad de la República). Recuperado de https://iie.fing.edu.uy/publicaciones/2011/DFF11

Decagon Devices América Latina Ltda. (2020). Recuperado el 1 de mayo de 2019, de Decagon Device website: http://www.decagon.com.br/

DeNA Co. Ltd. (2019). H2O - the optimized HTTP/2 server. Recuperado el 16 de mayo de 2020, de https://h2o.examp1e.net/index.html

Depari, A., Carvalho, D. F., Bellagente, P., Ferrari, P., Sisinni, E., Flammini, A., & Padovani, A.

(2019). An IoT based architecture for enhancing the effectiveness of prototype medical instruments applied to neurodegenerative disease diagnosis. *Sensors*, *19*(7), 19. https://doi.org/10.3390/s19071564

Devadiga, K. (2011). *IEEE 802.15.4 and the Internet of things*. 4–7. Recuperado de https://wiki.aalto.fi/download/attachments/59704179/devadiga-802-15-4-and-the-iot.pdf?version=1

Deveria, A. (2018). Can I use Websocket? Recuperado el 6 de marzo de 2020, de https://caniuse.com/#search=websocket

Deveria, A. (2020). Can I use HTTP/2? Recuperado el 16 de mayo de 2020, de https://caniuse.com/#search=http%2F2

Diaz Suárez, R. (2017). *Sistema seguro de monitoreo de variables utilizando redes de sensores inalámbricos (Tesis de maestría)*. (Universidad Internacional de la Rioja). Recuperado de https://reunir.unir.net/handle/123456789/5877

Dunkley, P., Llewellyn, G., Pascual, V., Salgueiro, G., & Ravindranath, R. (2016). The WebSocket Protocol as a Transport for the Message Session Relay Protocol (MSRP). *RFC 7977*, 1–28. Recuperado de https://tools.ietf.org/html/rfc7977

Eclipse Foundation. (2016). Jetty. Recuperado el 16 de mayo de 2020, de https://www.eclipse.org/jetty/about.html

Eclipse Foundation. (2018). Eclipse Mosquitto. Recuperado el 5 de marzo de 2020, de http://mosquitto.org/

Eclipse Foundation. (2019). Eclipse Californium™. Recuperado el 23 de noviembre de 2019, de https://www.eclipse.org/californium/

Eclipse Paho. (s/f). Eclipse Paho -MQTT and MQTT-SN software. Recuperado el 5 de marzo de

2020, de https://www.eclipse.org/paho/

Espinosa Gualotuña, C. F. (2015). *Adaptación del algoritmo OMEGA para ser utilizado en un prototipo de una red de sensores inalambrica 6LoWPAN (IPV6 over low power wireless personal area networks)(Tesis de pregrado).* (Escuela Politécnica Nacional). Recuperado de https://bibdigital.epn.edu.ec/handle/15000/11775

EVRYTHNG INC. (2016). Building the Web of Things. Recuperado el 6 de marzo de 2020, de https://apsg.bcs.org/materials/presentations/20160512.pdf

Fàbregas Bachs, M. (2016). *Diseño e implementación de una estación meteorológica de bajo coste con conectividad a Internet (Tesis de pregrado).* (Universitat Politècnica de Catalunya). Recuperado de https://upcommons.upc.edu/handle/2117/96723

Fallows, J., Ingham, D., & Godfrey, R. (Eds.). (2016). Advanced Message Queuing Protocol (AMQP) Websocket Binding (WSB) Version 1.0. *OASIS Committee Specification 01*, 1–18. Recuperado de http://docs.oasis-open.org/amqp-bindmap/amqp-wsb/v1.0/amqp-wsb-v1.0.pdf

Farkas, K., Nagy, A. Z., Tomas, T., & Szabo, R. (2014). Participatory sensing based real-time public transport information service. *IEEE International Conference on Pervasive Computing and Communication Workshops, PERCOM WORKSHOPS 2014*, 141–144. https://doi.org/10.1109/PerComW.2014.6815181

Fette, I., & Melnikov, A. (2011). The WebSocket Protocol. *RFC 6455*, 1–71. Recuperado de https://tools.ietf.org/html/rfc6455

Figueiredo, L. O., Lucas Maia, C. M., Rocha, M. T., Jose Barbosa Junior, A. N., Anna Aguiar, P. V. A., Rafael Lima, B. C., … Barros, P. R. (2018). Thermal vision for remote monitoring through cross-platform application. *13th IEEE International Conference on Industry*

Applications, INDUSCON 2018, 698–703. https://doi.org/10.1109/INDUSCON.2018.8627185

Fornaroli, M. F., Tugnarelli, M. D., Santana, S. R., & Díaz, J. (2018). Vulnerabilidades en HTTP/2. *XX Workshop de Investigadores en Ciencias de la Computación*, 1038–1042. Recuperado de http://sedici.unlp.edu.ar/handle/10915/68348

Fournaris, A. P., Giannoulis, S., & Koulamas, C. (2019). Evaluating CoAP end to end security for constrained wireless sensor networks. *2019 10th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2019 - Proceedings and Workshop*, 1–5. https://doi.org/10.1109/NTMS.2019.8763857

García Arano, C. (2010). *Impacto de la seguridad en redes inalámbricas de sensores IEEE 802.15.4 (Tesis de maestría)*. (Universidad Complutense Madrid). Recuperado de https://eprints.ucm.es/11312/1/Memoria_Fin_de_Master_-_Carlos_García_Arano.pdf

Gardašević, G., Veletić, M., Maletić, N., Vasiljević, D., Radusinović, I., Tomović, S., & Radonjić, M. (2017). The IoT Architectural Framework, Design Issues and Application Domains. *Wireless Personal Communications*, *92*(1), 127–148. https://doi.org/10.1007/s11277-016-3842-3

Gill Instruments - About us. (2020). Recuperado el 5 de enero de 2019, de Gill Instruments Limited website: http://gillinstruments.com/main/aboutusAN.html

Godfrey, R. (Ed.). (2018a). Advanced Message Queuing Protocol (AMQP) Enforcing Connection Uniqueness Version 1.0. *OASIS Committee Specification 01*, 1–12. Recuperado de http://docs.oasis-open.org/amqp/soleconn/v1.0/soleconn-v1.0.pdf

Godfrey, R. (Ed.). (2018b). Using the AMQP Anonymous Terminus for Message Routing Version 1 . 0. *OASIS Committee Specification 01*, (September), 1–19. Recuperado de

http://docs.oasis-open.org/amqp/anonterm/v1.0/anonterm-v1.0.pdf.%0Aanonterm-v1.0-cs01

Gonzalez, C., Flauzac, O., & Nolot, F. (2018). Evolución y Contribución para el Internet de las Cosas por las emergentes Redes Definidas por Software. *Memorias de Congresos UTP*, *1*(1), 28–33. Recuperado de http://revistas.utp.ac.pa/index.php/memoutp/article/view/1842

González García, A. J. (2017). *IoT : Dispositivos , tecnologías de transporte y aplicaciones (Tesis de maestría)*. (Universitat Oberta de Catalunya). Recuperado de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64286/3/agonzalezgarcia0TFM0617memoria.pdf

Gratzer, F. (2016). QUIC - Quick UDP Internet Connections. *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, (September), 39–46. https://doi.org/10.2313/NET-2016-09-1

Grigorik, I. (2013). HTTP/2. Recuperado el 15 de mayo de 2020, de High Performance Browser Networking website: https://hpbn.co/http2/

Grigorik, I. (2018). EM-WebSocket. Recuperado el 8 de marzo de 2020, de https://github.com/igrigorik/em-websocket

Grigorik, I. (2019). http-2: Pure Ruby implementation of HTTP/2 protocol. Recuperado el 16 de mayo de 2020, de https://github.com/igrigorik/http-2

Guo, L., Wu, J., Xia, Z., & Li, J. (2015). Proposed security mechanism for XMPP-based communications of ISO/IEC/IEEE 21451 sensor networks. *IEEE Sensors Journal*, *15*(5), 2577–2586. https://doi.org/10.1109/JSEN.2014.2373388

Hammond, D., & Lutsenko, N. (2018). SocketRocket. Recuperado el 6 de marzo de 2020, de https://github.com/facebook/SocketRocket

Hardie, T. (2016). Clarifying Registry Procedures for the WebSocket Subprotocol Name Registry. *RFC 7936*, 1–3. Recuperado de https://tools.ietf.org/html/rfc7936

Hartke, K. (2015). Observing Resources in the Constrained Application Protocol (CoAP). *RFC 7641*, 1–30. Recuperado de https://tools.ietf.org/html/rfc7641

Harz., A. C., Franco, E. D. la H., & Pinto, D. C. (2012). Las redes de sensores inalámbricos y el Internet de las cosas. *INGE CUC*, *8*(1), 163–172. Recuperado de http://revistascientificas.cuc.edu.co/index.php/ingecuc/article/view/253/232

Heidari, A. (2019). Artalk.Xmpp. Recuperado el 23 de mayo de 2020, de https://github.com/araditc/Artalk.Xmpp

Hermanudin, A. A., Ekadiyanto, F. A., & Sari, R. F. (2019). Performance Evaluation of CoAP Broker and Access Gateway Implementation on Wireless Sensor Network. *2018 IEEE Region 10 Symposium, Tensymp 2018*, 74–79. https://doi.org/10.1109/TENCONSpring.2018.8692050

HiveMQ. (2020). Enterprise ready MQTT to move your IoT data. Recuperado el 5 de marzo de 2020, de https://www.hivemq.com/

Home - Ecomatik Umweltmess- und Datentechnik. (2016). Recuperado el 1 de mayo de 2019, de Ecomatik website: https://ecomatik.de/en/

Hoz, J.David de, Saldana, J., Fernandez-Navajas, J., Ruiz-Mas, J., Rodriguez, R. G., & Luna, F. D. J. M. (2018). SSH as an Alternative to TLS in IoT Environments using HTTP. *2018 Global Internet of Things Summit (GIoTS)*, 1–6. https://doi.org/10.1109/GIOTS.2018.8534545

Huawei Cloud. (s/f). WebSocket Overview. Recuperado el 6 de marzo de 2020, de https://support.huaweicloud.com/en-us/api-sis/sis_03_0011.html

IEEE Std 802.15.4-2015. (2016). IEEE Standard for Low Rate Wireless Networks. *IEEE Standards Association*, 708. https://doi.org/10.1109/IEEESTD.2016.7460875

IETF. (2018). IETF | RFCs. Recuperado el 11 de agosto de 2018, de https://www.ietf.org/standards/rfcs/

Iglesias-Urkia, M., Orive, A., & Urbieta, A. (2017). Analysis of CoAP Implementations for Industrial Internet of Things: A Survey. *Procedia Computer Science*, *109*(2017), 188–195. https://doi.org/https://doi.org/10.1016/j.procs.2017.05.323

Imperva. (2016). HTTP / 2 : In-depth analysis of the top four flaws of the next generation web protocol. *The Imperva Hacker Intelligence Initiative*, 1–23. Recuperado de https://www.imperva.com/docs/Imperva_HII_HTTP2.pdf

Ishaq, I., Carels, D., Teklemariam, G., Hoebeke, J., Abeele, F., De Poorter, E., … Demeester, P. (2013). IETF Standardization in the Field of the Internet of Things (IoT): A Survey. *Journal of Sensor and Actuator Networks*, *2*(2), 235–287. https://doi.org/10.3390/jsan2020235

ISO/IEC 19464:2014. (2014). *Advanced Message Queuing Protocol (AMQP) v1.0 specification*. 124. Recuperado de https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

ISO/IEC 20922:2016. (2016). *Message Queuing Telemetry Transport (MQTT) v3.1.1*. 86. Recuperado de https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

Isode. (2020). M-Link XMPP Server. Recuperado el 23 de mayo de 2020, de https://isode.com/products/m-link.html

Järvinen, J., Marttinen, A., Jarvinen, R., Carlen, P., Luoma, M., Peuhkuri, M., & Manne, J. (2018). Enabling XEP-0258 Security Labels in XMPP. *IEEE Military Communications Conference MILCOM*, 661–666. https://doi.org/10.1109/MILCOM.2018.8599711

Kaazing. (s/f). About HTML5 WebSocket. Recuperado el 6 de marzo de 2020, de https://www.websocket.org/aboutwebsocket.html

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*, *3*(1), 1–7. Recuperado de https://pdfs.semanticscholar.org/ca6c/da8049b037a4a05d27d5be979767a5b802bd.pdf

Kayal, P., & Perros, H. (2017). A comparison of IoT application layer protocols through a smart parking implementation. *20th Conference on Innovations in Clouds, Internet and Networks, ICIN, Paris.*, 331–336. https://doi.org/10.1109/ICIN.2017.7899436

Kodali, Ravi K., & Valdas, A. (2019). MQTT implementation of IoT based fire alarm network. *International Conference On Communication, Computing and Internet of Things, IC3IoT 2018*, 143–146. https://doi.org/10.1109/IC3IoT.2018.8668158

Kodali, Ravi Kishore, Krishna Yogi, B. Y., Sharan Sai, G. N., & Honey Domma, J. (2018). Implementation of Home Automation Using CoAP. *TENCON 2018 - 2018 IEEE Region 10 Conference, Jeju, Korea (South)*, 1214–1218. https://doi.org/10.1109/TENCON.2018.8650135

Kodali, Ravi Kishore, & Sahu, A. (2016). An IoT based soil moisture monitoring on Losant platform. *2nd International Conference on Contemporary Computing and Informatics, IC3I 2016*, 764–768. https://doi.org/10.1109/IC3I.2016.7918063

Kodali, Ravi Kishore, & Sarjerao, B. S. (2017). A low cost smart irrigation system using MQTT protocol. *IEEE Region 10 Symposium (TENSYMP)*. https://doi.org/10.1109/TENCONSpring.2017.8070095

Konieczek, B., Rethfeldt, M., Golatowski, F., & Timmermann, D. (2015). Real-time

communication for the Internet of Things using jCoAP. *IEEE 18th International Symposium on Real-Time Distributed Computing, 2015*, 134–141. https://doi.org/10.1109/ISORC.2015.35

Kostromina, A., Siemens, E., & Babich, Y. (2018). A concept for a high-reliability meteorological monitoring system using AMQP. *International Conference on Applied Innovation in IT*, *6*(1), 109–115. https://doi.org/10.13142/kt10006.41

Krishna, C. S., & Sasikala, T. (2019). Healthcare Monitoring System Based on IoT Using AMQP Protocol. *International Conference on Computer Networks and Communication Technologies*, 305–319. https://doi.org/10.1007/978-981-10-8681-6_29

Kuladinithi, K., Bergmann, O., Thomas Pötsch, Becker, M., & Görg, C. (2011). *Implementation of coap and its application in transport logistics*. 1–7. Recuperado de http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf

Kumar, P., & Dezfouli, B. (2019). Implementation and analysis of QUIC for MQTT. *Computer Networks*, *150*, 28–45. https://doi.org/10.1016/j.comnet.2018.12.012

Lamy, O. (2020). eclipse/jetty.project. Recuperado el 16 de mayo de 2020, de https://github.com/eclipse/jetty.project

Latvakoski, J., & Heikkinen, J. (2019, octubre 8). A Trustworthy Communication Hub for Cyber-Physical Systems. https://doi.org/10.3390/fi11100211

Liang, Y., & Chen, Z. (2018). Intelligent and Real-Time Data Acquisition for Medical Monitoring in Smart Campus. *IEEE Access*, *6*, 74836–74846. https://doi.org/10.1109/ACCESS.2018.2883106

Libelium. (2018a). La nueva plataforma IoT para agricultura inteligente de Libelium proporciona la máxima precisión para la monitorización de cultivos. Recuperado el 19 de

junio de 2019, de Bussiness Wire website: https://www.businesswire.com/news/home/20180605006003/es/

Libelium. (2018b). Reducing Logistics' environmental impact by air quality monitoring in the Baltic Sea Port of Gdansk, Poland . Recuperado el 6 de marzo de 2020, de http://www.libelium.com/reducing-logistics-environmental-impact-by-air-quality-monitoring-in-the-baltic-sea-port-of-gdansk-poland/

Lima, J. C. de. (2018). *simulação de redes de sensores sem fio utilizando protocolos 6LOWPAN, RPL, MQTT e CoAP em smart cities (Tesis de maestria).* (Universidade de Passo Fundo). Recuperado de http://tede.upf.br:8080/jspui/handle/tede/1557

Liñán, A., Vives, A., Bagula, A., Zennaro, M., & Pietrosemoli, E. (2015). Internet De Las Cosas. En *ICTP* (Vol. 1). Recuperado de http://wireless.ictp.it/Papers/InternetdelasCosas.pdf

LINE Corporation. (2020). User manual — Armeria documentation. Recuperado el 16 de mayo de 2020, de https://line.github.io/armeria/docs/

Llamuca, E. S., Garcia, C. A., Naranjo, J. E., & Garcia, M. V. (2019). Cyber-physical production systems for industrial shop-floor integration based on AMQP. *International Conference on Information Systems and Software Technologies, ICI2ST 2019*, 48–54. https://doi.org/10.1109/ICI2ST.2019.00014

Londoño, D., González, M., Céspedes, S., Bustos, J., & Montenegro, G. (2017). Performance evaluation of HTTP/2 window size in the internet of things. *CEUR Workshop Proceedings*, *1950*(1), 31–33. Recuperado de http://repositorio.uchile.cl/bitstream/handle/2250/169015/item_85032722565.pdf?sequence=1

Loreto, S., Saint-Andre, P., Salsano, S., & Wilkins, G. (2011). Known Issues and Best Practices

for the Use of Long Polling and Streaming in Bidirectional HTTP. *RFC 6202*, 1–19. Recuperado de https://tools.ietf.org/html/rfc6202

Lv, Z., Meng, H., & Zhang, T. (2014). A cloud platform for distributed spectrum sensing. *Asia-Pacific Conference on Computer Aided System Engineering, APCASE 2014*, 1–4. https://doi.org/10.1109/APCASE.2014.6924462

Marín Pérez, J. (2018). *Análisis de prestaciones del protocolo HTTP2 (Tesis de pregrado).* (Universitat Politècnica de València). Recuperado de https://riunet.upv.es/handle/10251/107242

McManus, P. (2018). Bootstrapping WebSockets with HTTP/2. *RFC 8441*, 1–8. Recuperado de https://tools.ietf.org/html/rfc8441

Meera, M. S., & Rao, S. N. (2018). Comparative Analysis of IoT protocols for a Marine IoT System. *International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018*, 2049–2053. https://doi.org/10.1109/ICACCI.2018.8554906

Menchaca Paz, M. (2012). *Integración de ZigBee / 6LoWPAN en una red de sensores inalámbrica (Tesis de pregrado).* (Universidad de Cantabria). Recuperado de https://repositorio.unican.es/xmlui/bitstream/handle/10902/1464/349717.pdf?sequence=1&isAllowed=y

Merlino, G., Bruneo, D., Distefano, S., Longo, F., Puliafito, A., & Al-Anbuky, A. (2015). A smart city lighting case study on an OpenStack-powered infrastructure. *Sensors*, *15*(7), 16314–16335. https://doi.org/10.3390/s150716314

Mesa, L. F. (2019). Conozcamos sobre RabbitMQ, sus componentes y beneficios. Recuperado el 1 de marzo de 2020, de Academia Pragma website: https://www.pragma.com.co/academia/lecciones/conozcamos-sobre-rabbitmq-sus-

componentes-y-beneficios

Microsft Azure. (2020). What is Azure Service Bus? Recuperado el 2 de marzo de 2020, de https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview

Microsoft Azure. (2020). AMQP 1.0 in Azure Service Bus and Event Hubs protocol guide. Recuperado el 1 de marzo de 2020, de https://docs.microsoft.com/es-es/azure/service-bus-messaging/service-bus-amqp-protocol-guide

Mijovic, S., Shehu, E., & Buratti, C. (2016). Comparing application layer protocols for the Internet of Things via experimentation. *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow, RTSI , Bologna, 2016*, 1–5. https://doi.org/10.1109/RTSI.2016.7740559

Minerva, R., Biru, A., & Rotondi, D. (2015). Towards a definition of the Internet of Things (IoT). En *IEEE Internet of Things*. Recuperado de https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf

Montenegro, G., Cespedes, S., Loreto, S., & Simpson, R. (2016). *H2oT : HTTP / 2 for the Internet of Things*. 1–10. Recuperado de https://tools.ietf.org/id/draft-montenegro-httpbis-h2ot-00.htm

Montilla Pérez, A. (2018). *Pasarela sobre Raspeberry Pi entre intermediario MQTT y plataforma DBaas (Tesis de pregrado).* Universidad de Sevilla.

Moreno Cerdà, F. (2018). *Demostrador arquitectura publish / subscribe con MQTT* (Universitat Politècnica de Catalunya). Recuperado de http://hdl.handle.net/2117/117782

Morillo, A. (2017). *Evaluación del rendimiento en el nivel de red RPL en WSN (Tesis de*

pregrado). (Universidad de Sevilla). Recuperado de http://bibing.us.es/proyectos/abreproy/91091/fichero/TFG+ANTONIO+MORILLO+CANALEJO.pdf

Naik, N. (2017). Choice of Effective Messaging Protocols for IoT Systems : MQTT , CoAP , AMQP and HTTP. *IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017*, 1–7. https://doi.org/10.1109/SysEng.2017.8088251

Nathi, R. A., & Sutar, D. S. (2019). Embedded Payload Security Scheme using CoAP for IoT Device. *Proceedings - International Conference on Vision Towards Emerging Trends in Communication and Networking, ViTECoN 2019*, 1–6. https://doi.org/10.1109/ViTECoN.2019.8899549

Nolan, S. (2018). *Authenticated Payload Encryption Scheme for Internet of Things Systems over the MQTT Protocol (Tesis de maestría).* (The University of Dublin). Recuperado de https://www.scss.tcd.ie/publications/theses/diss/2018/TCD-SCSS-DISSERTATION-2018-003.pdf

Nottingham, M., & Nygren, E. (2018). The ORIGIN HTTP/2 Frame. *RFC 8336*, 1–11. Recuperado de https://tools.ietf.org/html/rfc8336

Novelli, L., Jorge, L., Melo, P., & Koscianski, A. (2018). Application Protocols and Wireless Communication for IoT: A Simulation Case Study Proposal. *11th International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2018*. https://doi.org/10.1109/CSNDSP.2018.8471765

Oracle. (2015). Package javax.websocket. Recuperado el 6 de marzo de 2020, de https://docs.oracle.com/javaee/7/api/javax/websocket/package-summary.html

Ortiz Mejía, E. L. (2018). *Desarrollo de un sistema prototipo de notificación temprana para*

personas menores de edad perdidas, en un ambiente cloud y usando el protocolo MQTT. (Tesis de pregrado) (Escuela Politécnica Nacional). Recuperado de https://bibdigital.epn.edu.ec/handle/15000/19642

Oryema, B., Kim, H. S., Li, W., & Park, J. T. (2017). Design and implementation of an interoperable messaging system for IoT healthcare services. *2017 14th IEEE Annual Consumer Communications and Networking Conference, CCNC 2017*, 45–51. https://doi.org/10.1109/CCNC.2017.7983080

Ozvural, G., & Kurt, G. K. (2015). Advanced approaches for wireless sensor network applications and cloud analytics. *IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2015*, (April), 1–5. https://doi.org/10.1109/ISSNIP.2015.7106979

paku. (2016). http2. Recuperado el 16 de mayo de 2020, de https://github.com/nekolunar/http2

Patierno, P. (2018). AMQP Essentials. Recuperado el 1 de marzo de 2020, de DZone website: https://dzone.com/refcardz/amqp-essentials?chapter=1

Pawara, S. R. (2017). Heterogeneous Health Monitoring System Using XMPP-Design and Implementation. *Proceedings of the 2017 2nd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2017*. https://doi.org/10.1109/ICECCT.2017.8117812

Peon, R., & Ruellan, H. (2015). HPACK: Header Compression for HTTP/2. *RFC 7541*, 1–55. Recuperado de https://tools.ietf.org/html/rfc7541

Pérez Leones, K. (2019). *Estudio y análisis del protocolo de mensajería avanzado en el internet de las cosas para aplicación en el campo de la domótica (Tesis de pregrado).* (Universidad católica de Santiago de Guayaquil). Recuperado de

http://repositorio.ucsg.edu.ec/handle/3317/13368

pgstath. (2016). Sharp.Xmpp . Recuperado el 23 de mayo de 2020, de https://github.com/pgstath/Sharp.Xmpp

Pierleoni, P., Pernini, L., Belli, A., Palma, L., Maurizi, L., & Valenti, S. (2016). Indoor localization system for AAL over IPv6 WSN. *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 1–7. https://doi.org/10.1109/PIMRC.2016.7794564

Pohl, M., Kubela, J., Bosse, S., & Turowski, K. (2018). Performance evaluation of application layer protocols for the internet-of-things. *Sixth International Conference on Enterprise Systems, ES 2018*, 180–187. https://doi.org/10.1109/ES.2018.00035

Povedano Molina, J. (2014). *Una aproximación Publicación/Subscripción centrada en datos para la provisión de servicios multimedia*. Universidad de Granada, Granada.

RabbitMQ by Pivotal. (2007). RabbitMQ Features. Recuperado el 2 de marzo de 2020, de https://www.rabbitmq.com/#features

Rahman, A., & Dijk, E. (2014). Group Communication for the Constrained Application Protocol (CoAP). *RFC 7390*, 1–46. https://doi.org/10.17487/RFC7390

Rahman, R. A., & Shah, B. (2016). Security analysis of IoT protocols: A focus in CoAP. *3rd MEC International Conference on Big Data and Smart City, ICBDSC 2016*, 172–178. https://doi.org/10.1109/ICBDSC.2016.7460363

Rajasekaran, P., Janardhan, R. P., & Chander, R. P. V. (2013). A smarter toll gate based on Web of Things. *IEEE International Conference on Electronics, Computing and Communication Technologies, CONECCT 2013*, 1–6. https://doi.org/10.1109/CONECCT.2013.6469318

Ramirez, J., & Pedraza, C. (2017). Performance analysis of communication protocols for

Internet of things platforms. *IEEE Colombian Conference on Communications and Computing (COLCOM)*, 1–7. https://doi.org/10.1109/ColComCon.2017.8088198

Rockliffe Systems. (2019). AstraChat | Hosted and On Premise Chat Solutions. Recuperado el 23 de mayo de 2020, de https://astrachat.com/

Rodas Vasquez, A., & Valencia Carrasquilla, A. (2018, agosto 9). Desarrollo e implementación de un prototipo para una plataforma tecnológica para la transmisión de texto y video (streaming) en tiempo real empleando tecnología websocket. Recuperado el 6 de marzo de 2020, de Ingenierías USBMed website: http://revistas.usbbog.edu.co/index.php/IngUSBmed/article/view/3277

Rosales Nuñez, S. A. (2015). *Control de acceso y seguridad inteligente (Tesis de Pregrado).* (Universidad de Sonora). Recuperado de http://www.repositorioinstitucional.uson.mx/bitstream/handle/unison/1688/rosalesnunezsergioalejandrol.pdf?sequence=1

Rose, Karen; Eldridge, Scott; Chapin, L. (2015). La internet de las cosas — Una breve reseña. *Internet Society*, 1–83. Recuperado de https://www.internetsociety.org/wp-content/uploads/2017/09/report-InternetOfThings-20160817-es-1.pdf

Roselin, A. G., Nanda, P., Nepal, S., He, X., & Wright, J. (2019). Exploiting the remote server access support of CoAP protocol. *IEEE Internet of Things Journal*, *6*(6), 1–1. https://doi.org/10.1109/jiot.2019.2942085

Rosero, P., Núñez, S., Realpe, S., Alvear, V., Beltrán, L., & Rosado, C. (2017). Internet de las cosas y redes de sensors inalámbricos: Review. *International Conference on Embedded Systems, (ICES 2017).*, 1–7. Recuperado de https://www.researchgate.net/publication/316438631_INTERNET_DE_LAS_COSAS_Y_

REDES_DE_SENSORES_INALAMBRICOS_REVIEW

Rubio Conde, P., Villarán Molina, D., & García Valls, M. (2017). Measuring performance of middleware technologies for medical systems: Ice vs AMQP. *ACM SIGBED Review*, *14*(2), 8–14. https://doi.org/10.1145/3076125.3076126

Ruz Mieres, D. V. (2019). *Evaluación del protocolo HTTP/2 para Internet de las Cosas (Tesis de pregrado).* (Universidad de Chile; Vol. 1). Recuperado de http://repositorio.uchile.cl/handle/2250/171099

Saint-Andre, P. (2004a). End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). *RFC 3923*, 1–27. Recuperado de https://tools.ietf.org/html/rfc3923

Saint-Andre, P. (2004b). Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM). *RFC 3922*, 1–34. Recuperado de https://tools.ietf.org/html/rfc3922

Saint-Andre, P. (2007). A Uniform Resource Name (URN) Namespace for Extensions to the Extensible Messaging and Presence Protocol (XMPP). *RFC 4854*, 1–9. Recuperado de https://tools.ietf.org/html/rfc4854

Saint-Andre, P. (2008). Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP). *RFC 5122*. Recuperado de https://tools.ietf.org/html/rfc5122

Saint-Andre, P. (2011a). Extensible Messaging and Presence Protocol (XMPP): Core. *RFC 6120*, 1–211. Recuperado de https://tools.ietf.org/html/rfc6120

Saint-Andre, P. (2011b). Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. *RFC 6121*, 1–114. Recuperado de

https://tools.ietf.org/html/rfc6121

Saint-Andre, P. (2015a). Extensible Messaging and Presence Protocol (XMPP): Address Format. *RFC 7622*, 1–27. Recuperado de https://tools.ietf.org/html/rfc7622

Saint-Andre, P. (2015b). Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP) This. *RFC 7590*, 1–9. Recuperado de https://tools.ietf.org/html/rfc7590b

Saint-Andre, P., Cridland, D., & Meijer, R. (2019). XEP-0001: XMPP Extension Protocols. Recuperado el 23 de mayo de 2020, de XMPP Standards Foundation website: https://xmpp.org/extensions/xep-0001.html

Saint-Andre, Peter. (2019). XEP-0199: XMPP Ping. Recuperado el 23 de mayo de 2020, de XMPP Standards Foundation website: https://xmpp.org/extensions/xep-0199.html

San José, S. G. (2015). *Implementación del Mecanismo de Acceso al Medio (MAC) IEEE 802.15.4e CSL (Coordinated Sampled Listening) sobre OpenWSN y Plataforma OpenMote (Tesis de maestría).* Universitat Oberta de Catalunya.

Saritha, S., & Sarasvathi, V. (2017). A study on application layer protocols used in IoT. *International Conference on Circuits, Controls, and Communications (CCUBE) 2017*, 155–159. https://doi.org/10.1109/ccube.2017.8394143

Sarjerao, B. S., & Prakasarao, A. (2018). A Low Cost Smart Pollution Measurement System Using REST API and ESP32. *2018 3rd International Conference for Convergence in Technology, I2CT 2018*, 1–5. https://doi.org/10.1109/I2CT.2018.8529500

Sauch Polo, V. (2017). *Red mallada de nodos Contiki para monitorización de temperaturas y envío de alertas por SMS (Tesis de pregrado).* (Universitat Jaumei). Recuperado de http://repositori.uji.es/xmlui/bitstream/handle/10234/174207/TFG_2017_Polo

Sauch_Victor.pdf?sequence=1&isAllowed=y

Schäfer, J. (2020). aioxmpp: An XMPP library for use with Python 3.4+ asyncio. Recuperado el 23 de mayo de 2020, de https://github.com/horazont/aioxmpp

Schreiber, F. V., Dias, R. A., Borba, B. De, Noll, V., & Gallo, V. Z. M. (2019). Application of the IoT paradigm for supervision in the utilities industry. *13th IEEE International Conference on Industry Applications, INDUSCON 2018*, 864–869. https://doi.org/10.1109/INDUSCON.2018.8627211

Sciarrone, H. N., & Sauro, G. B. (2017). *Sistema de adquisición y transmisión de señales biomédicas vía web, para monitoreo a distancia (Tesis de pregrado).* (Universidad Nacional de Mar del Plata). Recuperado de http://rinfi.fi.mdp.edu.ar/xmlui/handle/123456789/275

Selander, G., Mattsson, J., Palombini, F., & Seitz, L. (2019). Object Security for Constrained RESTful Environments (OSCORE). *RFC 8613*, 1–94. Recuperado de https://tools.ietf.org/html/rfc8613

Sergeyev, A. (2019). websocketd. Recuperado el 6 de marzo de 2020, de https://github.com/joewalnes/websocketd

Sethi, P., & Sarangi, S. R. (2017). Internet of Things : Architectures , Protocols , and Applications. *Journal of Electrical and Computer Engineering*, 25. https://doi.org/https://doi.org/10.1155/2017/9324035

Sharma, C., & Gondhi, N. K. (2018). Communication Protocol Stack for Constrained IoT Systems. *3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU) 2018*, 1–6. https://doi.org/10.1109/IoT-SIU.2018.8519904

Shelby, Z. (2012). Constrained RESTful Environments (CoRE) Link Format. *RFC 6690*, 1–22.

Recuperado de https://tools.ietf.org/html/rfc6690

Shelby, Z., Hartke, K., & Bormann, C. (2014). The Constrained Application Protocol (CoAP). *RFC 7252*, 1–112. Recuperado de https://tools.ietf.org/html/rfc7252

Shiva Shankar, J., S, P., S, C. V., & Sowmya, M. (2019). *MQTT in the Internet of Things*. 796–798. Recuperado de https://www.irjet.net/archives/V6/i12/IRJET-V6I12116.pdf

Shuang, K., Shan, X., Sheng, Z., & Zhu, C. (2014). An efficient zigbee-websocket based M2M environmental monitoring system. *12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014*, 322–326. https://doi.org/10.1109/DASC.2014.64

Silverlock, M. (2019). Gorilla Websocket. Recuperado el 6 de marzo de 2020, de https://github.com/gorilla/websocket

Simpson, R. (2019). Deuterium. Recuperado el 16 de mayo de 2020, de https://www.robbysimpson.com/deuterium/index.html

Soares de Souza, E. F., Melo Thiago, R., Azevedo, L. G., De Bayser, M., Torres da Silva, V., & Cerqueira, R. F. de G. (2018). Evaluation of Server Push Technologies for Scalable Client-Server Communication. *12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*, 1–10. https://doi.org/10.1109/SOSE.2018.00010

Sreeraj, S., & Kumar, G. S. (2018). Performance of IoT protocols under constrained network, a Use Case based approach. *International Conference On Communication, Computing and Internet of Things, IC3IoT 2018*, 495–498. https://doi.org/10.1109/IC3IoT.2018.8668105

Stenberg, D. (2014). HTTP2 explained. *ACM SIGCOMM Computer Communication Review*, *44*(3), 120–128. https://doi.org/10.1145/2656877.2656896

Sulaeman, A. B., Ekadiyanto, F. A., & Sari, R. F. (2016). Performance evaluation of HTTP-CoAP proxy for wireless sensor and actuator networks. *IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob) 2016*, 68–73. https://doi.org/10.1109/APWiMob.2016.7811451

Sultana, T., & Wahid, K. A. (2019). Choice of Application Layer Protocols for Next Generation Video Surveillance Using Internet of Video Things. *IEEE Access*, *7*, 41607–41624. https://doi.org/10.1109/ACCESS.2019.2907525

Syafarinda, Y., Akhadin, F., Fitri, Z. E., Yogiswara, Widiawanl, B., & Rosdiana, E. (2018). The Precision Agriculture Based on Wireless Sensor Network with MQTT Protocol. *IOP Conference Series: Earth and Environmental Science*, *207*(1). https://doi.org/10.1088/1755-1315/207/1/012059

Tajmajer, T., Sadkowski, T., & Winiecki, W. (2015). CoAP and database integration for sleeping and non-routable nodes. *IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) 2015*, *2*, 646–651. https://doi.org/10.1109/IDAACS.2015.7341383

Talaminos Barroso, A., Estudillo Valderrama, M. A., Roa, L. M., Reina Tosina, J., & Ortega Ruiz, F. (2016). A Machine-to-Machine protocol benchmark for eHealth applications - Use case: Respiratory rehabilitation. *Computer Methods and Programs in Biomedicine*, *129*, 1–11. https://doi.org/10.1016/j.cmpb.2016.03.004

Tandale, U., Momin, B., & Seetharam, D. P. (2017). An empirical study of application layer protocols for IoT. *International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, 2447–2451. https://doi.org/10.1109/ICECDS.2017.8389890

Tanyingyong, V., Olsson, R., Hidell, M., Sjödin, P., & Ahlgren, B. (2018). Implementation and Deployment of an Outdoor IoT-Based Air Quality Monitoring Testbed. *IEEE Global Communications Conference, GLOBECOM 2018*, 206–212. https://doi.org/10.1109/GLOCOM.2018.8647287

Teklemariam, G. K., Hoebeke, J., Van Den Abeele, F., Moerman, I., & Demeester, P. (2014). Simple RESTful sensor application development model using CoAP. *39th Annual IEEE Conference on Local Computer Networks Workshops 2014*, 552–556. https://doi.org/10.1109/LCNW.2014.6927702

TelosB. (s/f). Recuperado el 18 de febrero de 2020, de Consulting Measurement Technology website: http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/TelosB.html

Thantharate, A., Beard, C., & Kankariya, P. (2019). CoAP and MQTT based models to deliver software and security updates to IoT devices over the air. *International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 1065–1070. https://doi.org/10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00183

Thorson, P. (2020). websocketpp. Recuperado el 6 de marzo de 2020, de https://github.com/zaphoyd/websocketpp

TinyOS. (s/f). Recuperado el 1 de mayo de 2019, de http://www.tinyos.net/

Tsiatsis, V., Karnouskos, S., Höller, J., Boyle, D., & Mulligan, C. (2018). Architecture and State-of-the-Art. En *Internet of Things* (pp. 143–180). https://doi.org/10.1016/b978-0-12-814435-0.00019-5

Tsujikawa, T. (2020). nghttp2. Recuperado el 16 de mayo de 2020, de

https://github.com/nghttp2/nghttp2

Vadillo Gutiérrez, Ó. (2014). *Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN (Tesis de pregrado)*. (Universidad de Cantabria). Recuperado de http://repositorio.unican.es/xmlui/handle/10902/6023

Valiente Cristancho, A. (2017). *Integración de internet de las cosas en sistema embebido system on chip, con aplicación a domotica*. Universidad Distrital Franciasco José de Caldas.

Van den Abeele, F., Moerman, I., Demeester, P., & Hoebeke, J. (2017). Secure service proxy: A CoAP(s) intermediary for a securer and smarter web of things. *Sensors 2017*, *17*(7). https://doi.org/10.3390/s17071609

van der Stok, P., Bormann, C., & Sehgal, A. (2017). PATCH and FETCH Methods for the Constrained Application Protocol (CoAP). *RFC 8132*, 1–21. Recuperado de https://tools.ietf.org/html/rfc8132

Vardaro, M. J., Systems, H. I. T., AG, H. T., Jari, A., Pentti, M., Information, B. G., … Measurements, C. (2016). Constrained Application Protocol (CoAP) Option for No Server Response. *RFC 7967*, 1–18. Recuperado de https://tools.ietf.org/html/rfc7967

Vasseur, J. P. (2014). Terms Used in Routing for Low-Power and Lossy Networks. *RFC 7102*, 1–8. Recuperado de https://tools.ietf.org/html/rfc7102

Vera Pérez, J. (2017). *Análisis e implementación de redes inalámbricas de sensores deterministas robustas (Tesis de maestría)*. (Universitat Politècnica de València). Recuperado de https://riunet.upv.es:443/handle/10251/91814

Vergara, C., & Villan, M. O. (2017). El internet de las cosas (IoT) y la cuarta revolución industrial. Recuperado de energub smart energy website: https://energub.com/el-internet-de-las-cosas-iot-y-la-cuarta-revolucion-industrial/

Verma, P. (2017). Security of IoT data: Context, depth, and breadth across hadoop. En *Internet of Things and Data Analytics Handbook* (pp. 399–406). https://doi.org/10.1002/9781119173601.ch23

Vo, M. T., Do, N. H., Tran, V. S., Ma, Q. K., Le, C. T., & Mai, L. (2018). A multi-storey building actuator and sensor system using 6L0WPAN based Internet of Things: Practical design and implementation. *2nd International Conference on Recent Advances in Signal Processing, Telecommunications and Computing (SigTelCom) 2018*, 176–181. https://doi.org/10.1109/SIGTELCOM.2018.8325785

w3.css. (2020). libwebsockets.org. Recuperado el 6 de marzo de 2020, de https://libwebsockets.org/

W3C. (2012). The WebSocket API. Recuperado el 6 de marzo de 2020, de https://www.w3.org/TR/websockets/

Whited, S. (2020). Jackal: An XMPP server written in Go (Golang). Recuperado el 23 de mayo de 2020, de https://github.com/ortuman/jackal

Wikipedia. (2018). Reactor pattern. Recuperado el 6 de marzo de 2020, de https://en.wikipedia.org/wiki/Reactor_pattern

wolfSSL Inc. (2020). *wolfMQTT Client Library*. Recuperado de https://www.wolfssl.com/products/wolfmqtt/

XMPP Standards Foundation. (s/f-a). An Overview of XMPP. Recuperado el 14 de mayo de 2020, de XMPP Standards Foundation website: https://xmpp.org/about/technology-overview.html

XMPP Standards Foundation. (s/f-b). XMPP | Specifications. Recuperado el 23 de mayo de 2020, de XMPP Standards Foundation website: https://xmpp.org/extensions/

Yassein, M. B., Shatnawi, M. Q., & Al-Zoubi, D. (2016). Application layer protocols for the Internet of Things: A survey. *International Conference on Engineering and MIS (ICEMIS) 2016*, 1–4. https://doi.org/10.1109/ICEMIS.2016.7745303

Yoshino, T. (2015). Compression Extensions for WebSocket. *RFC 7692*, 1–28. Recuperado de https://tools.ietf.org/html/rfc7692

Yuan, M. (2017). Conociendo MQTT. *IBM Developer*. Recuperado de https://www.ibm.com/developerworks/ssa/library/iot-mqtt-why-good-for-iot/index.html

Zaphoyd Studios. (s/f). WebSocket++. Recuperado el 6 de marzo de 2020, de https://www.zaphoyd.com/websocketpp

Zikria, Y. Bin, Yu, H., Afzal, M. K., Rehmani, M. H., & Hahm, O. (2018). Internet of Things (IoT): Operating System, Applications and Protocols Design, and Validation Techniques. *Future Generation Computer Systems*, *88*, 699–706. Recuperado de https://www.sciencedirect.com/science/article/pii/S0167739X18317710