

# El protocolo MQTT

Este libro describe los aspectos fundamentales del funcionamiento del protocolo MQTT.

Además se da una pequeña guía de la librería `mqtt` de MicroPython utilizada en los dispositivos LoPy2 de Pycom<sup>©</sup>.

Sitio: [CampusVirtual](#)  
Curso: DISPOSITIVOS Y REDES INALÁMBRICOS  
Libro: El protocolo MQTT  
Imprimido por: ÁNGEL ORTEGA ALFARO  
Día: viernes, 1 de febrero de 2019, 21:53:53

## Tabla de contenidos

- [1. ¿Qué es MQTT?](#)
- [2. El modelo Publicar/Suscribir](#)
- [3. Temas \(topics\) - Direccionamiento MQTT](#)
- [4. El protocolo MQTT](#)
  - [4.1. Formato mensaje MQTT](#)
  - [4.2. Establecimiento conexión: Cliente-Broker](#)
  - [4.3. Publicación de un tema \(topic\)](#)
  - [4.4. Suscripción a un tema \(topic\).](#)
  - [4.5. QoS en MQTT](#)
- [5. Bróker MQTT](#)
  - [5.1. Bróker Eclipse Mosquitto](#)
- [6. Cliente MQTT](#)
  - [6.1. Cliente MQTT para Micro-Python.](#)
  - [6.2. Cliente MQTT en Node-Red](#)
- [7. Bibliografía](#)

## 1. ¿Qué es MQTT?

**MQTT** (*MQ Telemetry Transport*) es un protocolo para la **publicación de y subscripción a mensajes** ligero, abierto, simple y fácil de implementar. Sus características lo hacen ideal para comunicaciones máquina a máquina (M2M, Machine to Machine) y el IoT, donde los dispositivos tienen muy poca memoria para almacenar programas, por lo que se requiere una implementación con muy pocas líneas código, y en entornos de ancho de banda muy limitados.

## Historia.

MQTT fue inventado por Andy Stanford-Clark (IBM®) y Arlen Nipper (Arcom, actual Cirrus Link) en 1999, para ser utilizado como protocolo para comunicar los datos obtenidos en los oleoductos sobre una conexión vía satélite. Los requerimientos que el diseño inicial debía cumplir era: mínimo ancho de banda y un gasto energético muy reducido, para una mayor duración de las baterías de los dispositivos emisores.

El protocolo inicialmente desarrollado como un producto dentro del *middleware* IBM® MQ Series (de ahí el acrónimo MQ), fue liberado en 2010 y pasó a ser un estándar OASIS en 2014.

## Versiones.

En la actualidad existen varias versiones de MQTT:

- **MQTT v3.1**: Es la versión oficialmente aprobado por OASIS
- **MQTT v3.1.1**: Es una versión más reciente, pero con pocos cambios, de la MQTT v3.1. En la actualidad coexisten las dos por necesidades de compatibilidad, aunque se sugiere que siempre que se pueda se utilice la versión 3.1.1.
- MQTT v5: Es la última versión, todavía no aprobada (se espera su aprobación para enero de 2018). Debido a su incipiente estado el soporte para la misma es muy limitado. Un resumen sobre las líneas básicas de esta versión se puede encontrar [aquí](#).
- **MQTT-SN**: Especificada en 2013, es una versión del protocolo especialmente diseñada para redes de sensores (SN, *Sensor Network*), por lo que está optimizado para encapsularse en UDP, Zigbee u otros protocolos típicos de las redes de sensores.

En la actualidad las versiones de MQTT más utilizadas son la MQTT v3.1 y v3.1.1, también de ellas se dispone una mayor número de servidores desarrollados y de soporte. Ambas versiones encapsulan la mensajería sobre el protocolo TCP/IP. Aunque se espera una expansión de la versión MQTT-SN según se desarrolle más la tecnología IoT.

## 2. El modelo Publicar/Suscribir

MQTT se basa en un modelo publicar/suscribir de comunicación. Este modelo es una alternativa al clásico modelo cliente/servidor, en el que un dispositivo se pone directamente en comunicación con el otro destinatario de la comunicación con el que establece un diálogo.

En el **modelo publicar/suscribir** un dispositivo que quiere enviar un mensaje, **editor** (*publisher*), no conoce al destinatario del mismo, sino tan solo el tipo y contenido del mensaje a enviar. Del mismo modo, el consumidor (o destinatario final) del mensaje anterior, llamado **abonado** (*subscriber*), puede ser un solo dispositivo, o varios, tampoco conoce, ni se pone en comunicación con el dispositivo que generó el mensaje. Por contra, estos dispositivos abonados solo saben el tipo de mensaje y el contenido que quieren consumir.

El modelo necesita de un dispositivo intermedio llamado **bróker** cuya función es la recepción de los mensajes procedentes de un **editor**, filtrarlos y reenviarlos a los **abonados** correspondientes. Así pues tanto los editores como los abonados no solo interactúan con el bróker, y no entre sí.

En la [Figura 1](#), se puede ver un esquema del funcionamiento del modelo publicar/suscribir. Se puede ver como el editor envía el dato al bróker (publish: "21°C"), dicho dato es asociado a un **tema** (*topic*) temperature. Un **tema** es una cadena de caracteres con unas determinadas reglas que identifica el dato enviado por el editor, y que es utilizado por el bróker para filtrar los datos por él recibido. Además los temas son usados por los abonados como identificadores de los datos a los que se quieren suscribir.

Cuando un dato es recibido por el bróker, este lo filtra en función de su tema, y lo redirige a todos los abonados que están suscritos, mensaje *subscribe*, a dicho tema. El envío por parte del bróker a los abonados se realiza mediante un mensaje *publish*: "21°C", similar al que envió el editor al broker.

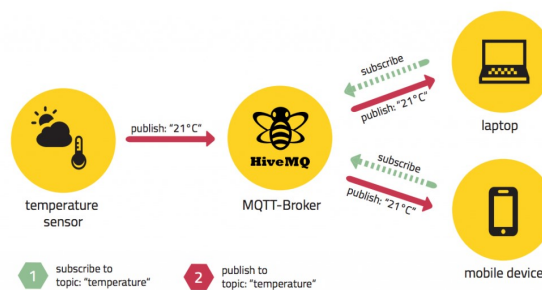


Figura 1. Modelo Publicar/Suscribir. (Figura obtenida de [www.hivemq.com](http://www.hivemq.com))

### Desacoplamiento editor/abonado

Como se puede observar tanto los abonados como los editores están totalmente desacoplados uno de otros, es decir, que no tienen ninguna relación entre ellos. Dicho desacoplamiento se da en tres sentidos:

- **Desacoplamiento espacial:** los editores y abonados no saben de la existencia del otro, no conocen ni su IP ni el puerto de la aplicación) del otro.
- **Desacoplamiento temporal:** Ambos editores y abonados no necesitan estar ejecutándose en el mismo instante de tiempo
- **Desacoplamiento de sincronización:** durante la publicación o recepción de mensajes, por un editor o abonado, no se detienen las operaciones en el otro dispositivo (abonado o editor).

### 3. Temas (topics) - Direccionamiento MQTT

Los **temas** (en inglés *topics*) con el mecanismo de direccionamiento del protocolo MQTT. Los editores comunican información a los abonados porque aquellos publican datos en un tema al que éstos están suscritos.

En MQTT **no hay una definición formal previa de los temas** que se pueden publicar o a los que abonarse. En este sentido, los editores tiene total libertad para definir cual el identificador y el contenido de cada tema que ellos publican, así como el instante en que realizan la publicación.

Evidentemente el identificado y estructura del tema deberá ser conocido por el abonado para poder obtener e interpretar de manera correcta la información que publicó el editor.

#### Formato de los temas (*topics*).

Un tema o *topic* no es más que una cadena de caracteres en formato UTF-8, que cumple las siguientes reglas:

- Debe tener al menos un carácter de longitud.
- Es sensible a mayúsculas y minúsculas
- Puede incluir espacios en blanco.
- Se puede utilizar el carácter "/" para organizar los temas (*topics*) en niveles.
- Un tema (*topic*) con un solo carácter "/" es válido.
- Un tema (*topic*) no debe de contener el caracter nulo (uTF-8: 0xC0 ó 0x80; Unicode U+0000)
- Al estar representado en UTF-8, la máxima longitud de un tema (*topic*) es de 64 Kbytes (65535 bytes).

Algunos ejemplos se muestran a continuación:

- casa/comedor/iluminacion/lampara
- casa/comedor/iluminacion/Lampara
- casa/comedor/electrodomesticos/DVD
- casa/comedor/electrodomesticos/TV
- casa/alcoba/iluminacion/lampara
- casa/alcoba/iluminacion/mesita drcha
- USA/New Jersey/trafficLight/125468
- 5ff4a2ce-e485-40f4-826c-b1a5d81be9b6/status

Como se puede observar, los nombres de los temas o *topics* se pueden interpretar como si fueran nombres de archivos en el sistema operativo Linux. De esta forma los caracteres "/" representan distintos niveles (directorios en el caso de archivos Linux) en los que organizar los temas. Así podemos decir que: el tema `casa/comedor/iluminación` engloba a dos *topics* `casa/comedor/iluminacion/lampara` y `casa/comedor/iluminacion/Lampara`. Podemos ver que estos dos últimos son temas distintos por la distinción entre mayúsculas y minúsculas.

Esta forma de organización de los temas o *topics* hace posible el empleo de **comodines** (*wildcard*) para especificar más de un tema (*topic*) con un solo nombre de tema. A continuación vemos como se utilizan.

#### Comodines o *wildcards*.

Los **comodines** (*wildcards*, en inglés) son utilizados a la hora de suscribirse a temas (*topics*) para especificar con un solo nombre de tema (*topic*) más de un tema (*topic*) al que suscribirse. Existen dos *wildcards* o comodines:

- "+": Se puede interpretar de manera similar al carácter "?" en el caso de archivos en Linux. Este símbolo se podrá sustituir por cualquier cadena de caracteres dentro del nivel donde es especificado.

Así si un abonado se suscribe al tema (*topic*) `casa/planta 1/+/temperatura` estará suscribiéndose a todos los siguientes *topics* o temas:

- casa/planta 1/comedor/temperatura
- casa/planta 1/cocina/temperatura
- casa/planta 1/water/temperatura

pero no estará suscrito a los *topics*:

- casa/escaleras/temperatura
- casa/planta 2/habitacion niños/temperatura
- casa/planta 1/pasillo/luminosidad

- "#": Este comodín se puede interpretar como el comodín "\*" en el caso de archivos en Linux. Pero este comodín solo puede utilizarse al final del nombre de un tema o *topic*.

Así siguiendo con el ejemplo anterior el *topic* `casa/planta 1/#` se referirá a todos los temas que comiencen con la cadena `"casa/planta 1/"` que en nuestro caso serán:

- `casa/planta 1/comedor/temperatura`
- `casa/planta 1/cocina/temperatura`
- `casa/planta 1/water/temperatura`
- `casa/planta 1/pasillo/luminosidad`

Dado que el carácter "#" se interpreta como cualquier cadena a partir de su posición si especificamos el tema (*topic*) "#", es decir, **solo el caracter "#"**: esto se referirá a todos los temas o *topics* almacenados en el bróker, por lo que con una sola petición de suscripción podríamos suscribirnos a todos los temas (*topics*) que maneja el bróker.

## 4. El protocolo MQTT

Como ya se ha comentado el protocolo MQTT se basa en el desacoplamiento entre clientes, basándose en el modelo de comunicación publicar/suscribir. En este modelo un cliente (editor) publica un determinado mensaje para que otros clientes (abonados) puedan recibirlos. Como la comunicación no es directa entre editores y abonados, se necesita de un elemento que se encargue de la redistribución de dichos mensajes es el llamado bróker.

De esta manera se puede hablar de dos tipos de elementos en la comunicación MQTT: los clientes y el bróker.

### Cientes MQTT.

Un cliente MQTT es cualquier editor y/o abonado que publica o se suscribe a un tema (*topic*). Una de las características fundamentales de los clientes MQTT es que, sobre todo en el caso de los editores, suelen ser micro-controladores totalmente autónomos, con poca capacidad de memoria y cálculo, y alimentación mediante baterías. Por ello la implementación del cliente debe necesitar muy pocos recursos de memoria, y también deberá de consumir poca energía.

Hay que tener en cuenta que cualquier cliente MQTT puede ser simultáneamente editor y abonado. Pudiéndose abonar a uno o más *topics*, a la vez que puede publicar uno o más temas (*topics*); no teniendo por qué ser los temas publicados idénticos a los temas a los que se abona.

Actualmente existen un gran número de implementaciones de clientes MQTT en una gran cantidad de lenguajes de programación.

### Bróker MQTT.

El bróker será el dispositivo que da soporte al modelo editar/suscribir en que se basa MQTT. Es fundamental ya que él se encarga de recibir todos los mensajes publicados por los editores, filtrarlos, y determinar que abonados están interesados en qué mensajes, reenviándolos.

Evidentemente el bróker deberá de mantener todas las conexiones con los editores y abonados activos, para así determinar que suscripciones están activas, y deberá reenviar, y cuales no, descartando los mensajes publicados para dichas suscripciones no activas.

Desde el punto de vista clásico del modelo cliente servidor, el bróker sería el servidor y los editores y abonados serían los clientes.

### Características generales del protocolo.

El protocolo **MQTT v3.1.1** se basa en el protocolo **TCP**, siendo el puerto en el que el bróker se pone a la escucha de nuevas conexiones el **puerto 1883**; en el caso de utilizar **SSL** el puerto sería el **8883**. Cómo en todo modelo cliente/servidor, en el caso de MQTT es el cliente, ya sea abonado o editor, el que deberá de establecer una conexión con el bróker determinado. De lo que se deduce que todo cliente debe conocer la IP del bróker al que se conectará.

Desde el punto de vista del cliente MQTT, el bróker es el destinatario final de las publicaciones, y el origen de los mensajes a los que está abonado, ya que él no conoce la existencia de los otros clientes que son los que realizan las otras funciones de editor y/o abonado.

Dado que toda conexión es iniciada por un cliente (editor/abonado), el que la conexión se deba realiza a través de un NAT no supondrá ningún problema. Por lo que los clientes podrán tener un direccionamiento IP privado.

MQTT utiliza una **comunicación basada en comando/confirmación**, teniendo todo comando una confirmación asociada.

## 4.1. Formato mensaje MQTT

El formato de un mensaje MQTT dependerá del tipo de mensaje de que se trate, aunque todos ellos tienen una **cabecera fija**. Después de la cual hay mensajes que requieren una **cabecera variable adicional** y un **payload** o carga útil. El formato general de un mensaje MQTT se puede observar en la [Figura 1](#).

Cabecera Fija. Presente en todos los mensajes MQTT
Cabecera Variable. Presente en algunos mensajes MQTT
Payload. Presente en algunos mensajes MQTT

Figura 1.- Formato general de un mensaje MQTT.

### 1.- Formato para la Cabecera fija.

Todos los mensajes del protocolo MQTT comienzan con una cabecera fija, cuyo contenido se muestra en la [Figura 2](#)

	7	6	5	4	3	2	1	0
Byte 1	Tipo Mensaje				Flag DUP	QoS		RETAIN
Byte 2	Longitud Restante							
.....								
Byte 4								

Figura 2.- Formato de la cabecera fija de un mensaje MQTT.

La longitud total de la cabecera fija de un mensaje MQTT va desde un mínimo de 2 bytes hasta un máximo de 4, dependiendo del tamaño del campo **Longitud Restante**. El significado de cada uno de los campos se detalla a continuación:

- **Tipo Mensaje (4 bits)**: Indica mediante un código de 4 bits el tipo de mensaje de que se trata. En la [Tabla 1](#) se detalla el significado de cada uno de los valores que puede tomar este campo.

Tabla 1.- Valor del campo "Tipo de Mensaje" de la cabecera fija de un mensaje MQTT.

Mnemotécnico	Valor (hex)	Descripción
Reservado	0x0	Reservado
CONNECT	0x1	Petición de conexión por parte del cliente al bróker.
CONNACK	0x2	Confirmación de conexión (del bróker al cliente).
PUBLISH	0x3	Publicación de un mensaje.
PUBACK	0x4	Confirmación de recepción de publicación.
PUBREC	0x5	Publicación asegurada recibida. Respuesta a un PUBACK con QoS=2
PUBREL	0x6	Publicación asegurada publicada. Respuesta a un PUBREC.
PUBCOMP	0x7	Publicación asegurada completada. Respuesta a un PUBREL.
SUBSCRIBE	0x8	Petición de suscripción por parte del cliente al bróker.
SUBACK	0x9	Confirmación de suscripción (del bróker al cliente).
UNSUBSCRIBE	0x10	Petición de cese de suscripción del cliente al bróker.
UNSUBACK	0x11	Confirmación de cese de suscripción (del bróker al cliente).
PINGREQ	0x12	Petición de PING.
PINGRESP	0x13	Respuesta a una petición de PING.



DISCONNECT	0x14	El cliente se desconectará.
Reservado	0x15	Reservado

- **Flag DUP (1 bit):** Flag utilizado para comunicar que el mensaje que se envía es un mensaje duplicado, es decir, igual a uno previamente enviado. En el caso de enviar mensajes con un valor de QoS mayor a cero (0), dichos mensajes deben ser confirmados, y en el caso de no confirmarse se deben de reenviar. Es cuando se reenvían los mensajes cuando el emisor (ya sea cliente o bróker) debe de poner este bit a 1 para indicar que el mensaje ya se envió previamente.

Si el Flag DUP = 1 en la cabecera variable debe incluirse el campo Mensaje\_ID, para relacionar le mensaje actual con su duplicado, identificado por el valor del campo Mensaje\_ID.

- **QoS (2 bits):** Indica el nivel de seguridad en la entrega a emplear para un mensaje PUBLISH. Su valor e interpretación se muestra en la [Tabla 2](#).

Tabla 2.- Valor del campo "QoS" de la cabecera fija de un mensaje MQTT.

Nivel QoS	Valor(hex)	Descripción
0	0x0	"At most once": En este nivel la publicación es enviada y no se requiere confirmación de recepción.
1	0x1	"At least once": En este nivel de servicio la publicación es enviada y debe ser confirmada por el receptor. Si no se confirma se volverá a enviar. Por lo que el mensaje será recibido por el receptor al menos una vez.
2	0x2	"Exactly once": En este nivel de servicio se asegura la entrega del mensaje en recepción sólo una vez. Evitándose la existencia de duplicados.
3	0x3	Reservado

El comportamiento de cada uno de los niveles de QoS se verá en la sección "QoS en MQTT".

- **RETAIN (1 bit):** Este campo solo es utilizado en mensajes PUBLISH. Cuando un cliente envía un mensaje PUBLISH al bróker con el bit RETAIN=1, el bróker deberá guardar el contenido de dicho mensaje tras su envío a todos los abonados al mismo *topic*. El objetivo de esta retención del mensaje publicado es que cuando una nueva suscripción sea realizada al *topic* anterior, el bróker enviará directamente el valor almacenado al abonado, sin esperar a recibir una nueva publicación sobre dicho *topic*.
- **Longitud Restante (de 1 a 4 bytes):** Indica el número de bytes que se incluyen en el mensaje sin contar los bytes de la cabecera fija, pero sí los de la cabecera variable y el *payload* si existieran. Si se utiliza un sólo byte se podrán codificar mensajes de longitud de hasta 127 bytes, aunque se puede llegar a codificar longitudes de hasta 256 MB si se utilizan los 4 bytes disponibles.

Cada byte puede codificar 128 valores utilizando los 7 bits menos significativos y dejando el bit 8 como "**bit de continuación**" el cual indica si se ha de tomar el siguiente byte para seguir codificando la longitud del mensaje o no. Así pues el valor codificado viene dado por la [ecuación 1](#):

$$\text{longitud\_codificada} = \text{valor}_0 + b_0 * (\text{valor}_1 * \text{base} + b_1 * (\text{valor}_2 * \text{base}^2 + b_2 * \text{valor}_3 * \text{base}^3)) \quad (1)$$

donde:

- $\text{base} = 128$ , base en la que se codifica la longitud.
- $\text{valor}_i$  = es el valor decimal codificado en los 7 bits menos significativos del byte  $i$ .
- $b_i$  = es el valor del bit más significativo del byte  $i$ . Y que indica si se ha de tomar el siguiente byte o no para seguir codificando la longitud.

Como se puede comprobar lo que se hace es una descomposición polinómica en base 128 del valor de la longitud.

## 2.- Formato para la Cabecera Variable.

Algunos mensajes MQTT requieren de la utilización de una cabecera variable, la cual se sitúa entre la cabecera fija, anteriormente descrita, y el *payload*. El contenido de la cabecera variable está en función del mensaje MQTT que la contiene. Una descripción de esta cabecera se realizará más adelante cuando se describan los mensajes que la requieren.

## 3.- Formato para el *payload*.

No todos los mensajes MQTT contienen una parte de *payload*, dependiendo por lo tanto su contenido del tipo de mensaje que lo contiene. En la [Tabla 3](#) se muestra una lista de los mensajes MQTT que contienen un campo de *payload*.

Tabla 3.- Mensajes MQTT con un campo *payload*.

Tipo de Mensaje	<i>Payload</i>
CONNECT	Obligatorio
CONNACK	NO Presente
PUBLISH	Opcional
PUBACK	NO Presente
PUBREC	NO Presente
PUBREL	NO Presente
PUBCOMP	NO Presente
SUBSCRIBE	Obligatorio
SUBACK	Obligatorio
UNSUBSCRIBE	Obligatorio
UNSUBACK	NO Presente
PINGREQ	NO Presente
PINGRESP	NO Presente
DISCONNECT	NO Presente

## 4.2. Establecimiento conexión: Cliente-Broker

Al necesitar MQTT soporte TCP para su funcionamiento se ha de establecer una conexión entre el cliente y el bróker antes de poder realizar ningún intercambio de mensajes.

El establecimiento de conexión es comenzado por el cliente enviando un mensaje CONNECT al bróker. El cual contestará con un mensaje CONNACK indicando que la conexión ha se estableció. Una vez la conexión está abierta el bróker es el encargado de mantenerla abierta mientras no reciba un mensaje DISCONNECT por parte del cliente, o se detecte una pérdida de conectividad. Un esquema de la apertura de la conexión se puede ver en la [Figura 1](#).

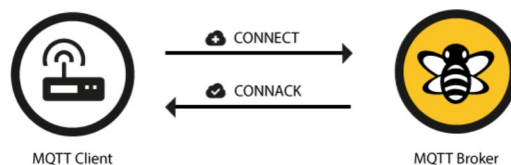


Figura 1.- Establecimiento de una conexión MQTT. (Figura obtenida de [www.hivemq.com](http://www.hivemq.com))

A continuación se describirá el contenido de los mensajes implicados en la comunicación.

### 1.- Formato mensaje CONNECT.

Un esquema del mensaje CONNECT es mostrado en la [Figura 2](#), donde el valor **Tipo Mensaje = 0x1**

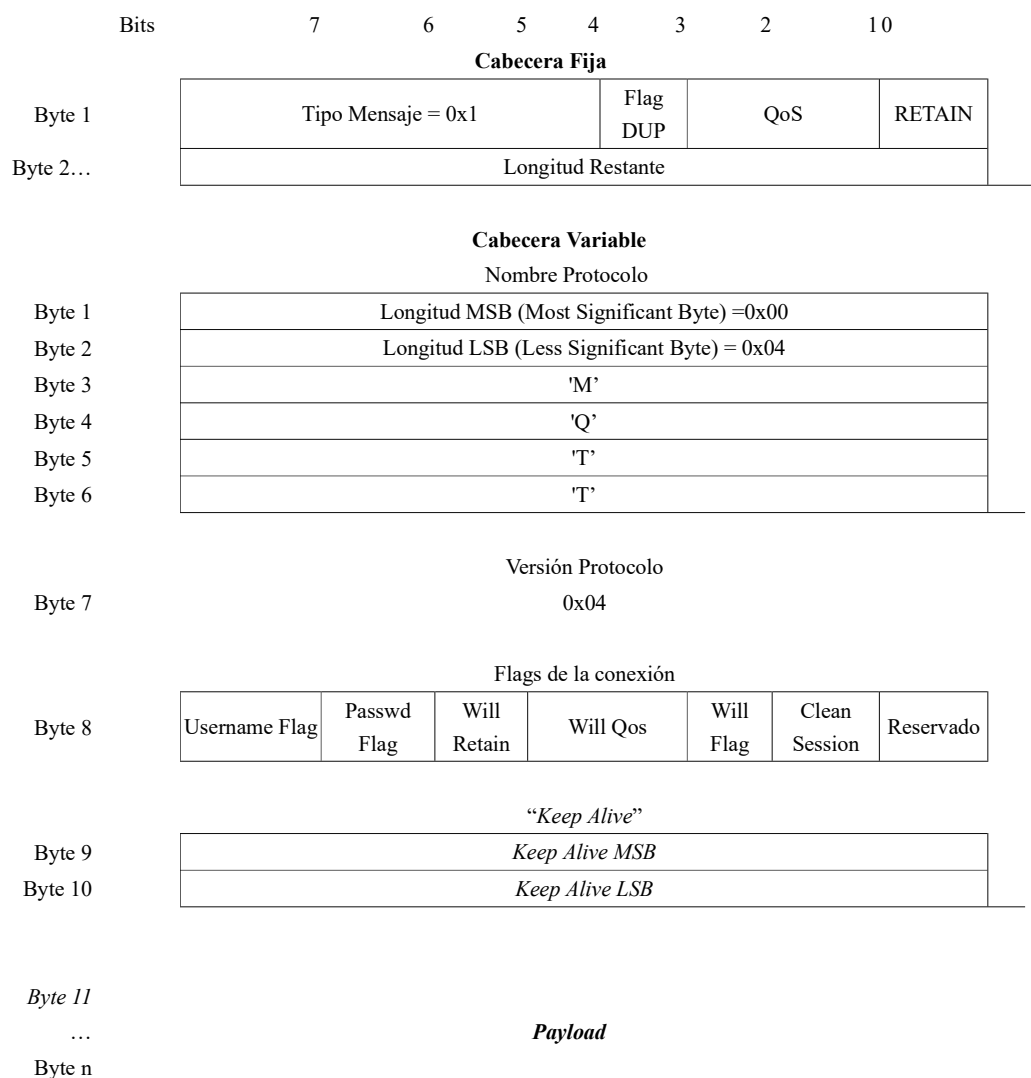


Figura 2.- Formato de un mensaje CONNECT.

En la [Figura 2](#), se puede ver como el mensaje CONNECT está formado por tres partes: la cabecera fija, ya descrita en la sección "[Formato mensaje MQTT](#)", la cabecera variable, y el *payload*. A continuación pasamos a describir el contenido de cada una de estas dos últimas partes.

## 1.1.- Contenido de la Cabecera Variable.

El contenido de la cabecera variable consta de cuatro campos principales: Nombre Protocolo, Versión Protocolo, Flags de la conexión y "Keep Alive" que pasamos a describir a continuación:

- **Nombre Protocolo (6 bytes):** Conjunto de 6 bytes que se emplean para indicar el nombre del protocolo a utilizar. Este campo está dividido en los siguientes subcampos
  - **Longitud MSB (1 byte):** Byte más significativo del valor total que indica la longitud en bytes del nombre del protocolo a utilizar.
  - **Longitud LSB (1 byte):** Byte menos significativo del valor total que indica la longitud en bytes del nombre del protocolo a utilizar.
  - **Nombre del protocolo (número variable de bytes):** Cadena de caracteres de longitud indicada en los dos bytes anteriores, y que contiene el nombre del protocolo. Según la versión del protocolo el mensaje será:
    - **Versión 3.1:** En este caso el nombre será "MQIsdp", siendo el valor de MSB=0x00 y de LSB = 0x06.
    - **Versión 3.1.1:** Es el caso representado en la [Figura 2](#), y dicho nombre viene dado por la cadena de caracteres "MQTT", de ahí que los valores Longitud MSB = 0x00 y el valor Longitud LSB = 0x04. Lo que indica que el nombre del protocolo está compuesto por cuatro caracteres.
- **Versión Protocolo (1 bytes):** Indica con una valor binario la versión del protocolo, para el caso que nos ocupa MQTT v3.1.1, el campo tomará el valor de 0x04, ya que a esta versión se la numera con el valor 4. Para MQTT v3.1 el valor del campo será 0x03.
- **Flags de la conexión (1 byte):** Es un bytes de *flags* que determinan entre otras cosas el contenido del *Payload*. Los *flags* transmitidos son:
  - **Username Flag (1 bit):** Indica si se incluye en el *Payload* un campo con el nombre de usuario.
  - **Passwd Flag (1 bit):** Indica si se incluye en el *Payload* la clave asociada al nombre de usuario anterior. Estos dos campos son utilizados para autenticación por parte del bróker.
  - **Will Retain (1 bit):** Cuando su valor es 1 el bróker deberá de mantener almacenado en su memoria el "Will Message" tras su transmisión.

¿Qué es el "Will Message"? Este es un mensaje que el bróker enviará cuando detecte un cierre abrupto de la conexión que se establece con el mensaje CONNECT. Si el cierre de la conexión se realiza de manera ordenada dicho mensaje no se emitirá

- **Will QoS (2 bits):** Es el nivel de QoS que se aplicará al "Will Message" cuando este se emita.
- **Will Flag (1 bit):** Indica si se utilizará un "Will Message" para esta conexión o no. En el caso de tener este *flag* a 0, los *flags*: *Will Retain* y *Will QoS* no tienen ningún efecto.
- **Clean Session:** En el caso de tener este *flag* a 1, de haber existido una sesión previa con el mismo cliente, tanto el bróker como el cliente deberán descartar cualquier información sobre la misma. En el caso de un valor de *flag* igual a 0, el servidor debe utilizar la información almacenada de una sesión para el establecimiento de esta nueva sesión.
- **"Keep Alive":** Indica mediante dos bytes el número máximo de segundos que deben de pasar entre dos mensajes enviados por el cliente. El cliente será el responsable de enviar un mensaje al bróker antes de que pasa dicho intervalo de tiempo. En el caso de no tener ningún otro mensaje que enviar antes de la expiración del tiempo el cliente deberá enviar un mensaje PINGREQ al bróker para cumplir con esta restricción.

## 1.2.- Contenido del Payload.

Como se ha comentado en la sección anterior, el contenido del *Payload* dependerá entre otras cosas del valor de del campo *Flags* de la conexión. Aunque su esquema general es el mostrado en la [Figura 3](#).

7	6	5	4	3	2	1
Cliente_ID						
Will Topic						
Will Message						
Nombre Usuario						
Clave Usuario						

Figura 3.- Contenido del *Payload* de un mensaje CONNECT.

donde los campos tienen el siguiente contenido:

- **Cliente\_ID(entre 1 y 23 bytes):** es un campo obligatorio que indica en UTF-8 un identificador del cliente (editor/abonado). La especificación de este campo se puede ver en la [Figura 4](#).

	7	6	5	4	3	2	1
Byte 1	Longitud MSB						Longitud LSB
Byte 2							
Byte 3	Cadena con el Cliente_ID						
...							
Byte 23							

Figura 4.- Contenido del campo Cliente\_ID del *Payload*.

Como se puede ver en la [Figura 4](#), los dos primeros bytes del campo `Cliente_ID` indican la longitud total de la cadena de caracteres que contiene el identificador del cliente. Estos dos bytes van seguidos de la cadena de caracteres con el identificador del cliente en sí, y cuya longitud se ha indicado en los dos bytes anteriores.

- **Will Topic (número de bytes variable):** Su aparición dependerá del valor del *Will Flag*. Y en caso de aparecer contendrá una cadena de caracteres codificada en UTF-8 que indicará el *topic* o tema del *Will Message*. El formato de este campo es similar al del campo `Cliente_ID` descrito anteriormente, pero en este caso la cadena de caracteres tendrá el formato especificado en la sección "[Temas \(topics\) - Direccionamiento MQTT](#)".
- **Will Message (número de bytes variable):** Campo opcional, aparecerá si el *Will Flag*= 1. Contendrá el cuerpo del mensaje asociado al *topic* indicado en el campo anterior (*Will Topic*). El formato es similar al mostrado en el campo `Cliente_ID`, solo que ahora después de los dos primeros bytes que indican la longitud del *Will Message*, los bytes sucesivos codificarán en binario el contenido del mensaje. Dicho contenido es binario ya que el mensaje puede contener cualquier estructura de datos, la cual deberá ser conocida por el receptor del mensaje (el bróker) para poder ser interpretada.
- **Nombre Usuarios (número de bytes variable):** Este campo opcional aparece si el *Username Flag* es inicializado a 1. Su contenido es similar al del campo *Will Topic*, solo que ahora se codifica el nombre de un usuario. Esta información será utilizada por el bróker para funciones de autenticación y autorización.
- **Clave Usuario (número de bytes variable):** Campo opcional que aparece si el *Passwd Flag* es inicializado a 1 en la cabecera variable. Tiene un formato similar al campo *Will Topic*, solo que codificará en binario, no en UTF-8, la clave de usuario asociada al nombre de usuario del campo anterior. Ambos campos pueden ser usado por el bróker para funciones de autenticación y autorización.

### 1.3 Ejemplo de captura

Un ejemplo de un mensaje CONNECT capturado mediante la herramienta wireshark es mostrado a continuación:

```
Frame 548: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
Ethernet II, Src: Espressi_00:f1:3e (24:0a:c4:00:f1:3e), Dst: Luxshare_00:2b:c2 (3c:18:a0:00:2b:c2)
Internet Protocol Version 4, Src: 192.168.10.9, Dst: 192.168.10.8
Transmission Control Protocol, Src Port: 53115, Dst Port: 1883, Seq: 13, Ack: 1, Len: 22
[2 Reassembled TCP Segments (34 bytes): #546(12), #548(22)]
MQ Telemetry Transport Protocol
Connect Command
0001 0000 = Header Flags: 0x10 (Connect Command)
0001 .... = Message Type: Connect Command (1)
.... 0... = DUP Flag: Not set
.... .00. = QOS Level: Fire and Forget (0)
.... ...0 = Retain: Not set
Msg Len: 32
Protocol Name: MQTT
Version: 4
1100 0010 = Connect Flags: 0xc2
1... .... = User Name Flag: Set
.1.. .... = Password Flag: Set
..0. .... = Will Retain: Not set
...0 0... = QOS Level: Fire and Forget (0)
.... .0.. = Will Flag: Not set
.... ..1. = Clean Session Flag: Set
.... ...0 = (Reserved): Not set
Keep Alive: 0
Client ID: Pycom1
User Name: pycom
Password: pycom
```

donde en verde se muestra el contenido de la cabecera fija, en naranja el de la cabecera variable, y el *payload* se resalta en rojo.

## 2. Formato mensaje CONNACK .

El formato de un mensaje CONNACK se muestra en la [Figura 5](#).

Bits	7	6	5	4	3	2	10
Cabecera Fija							

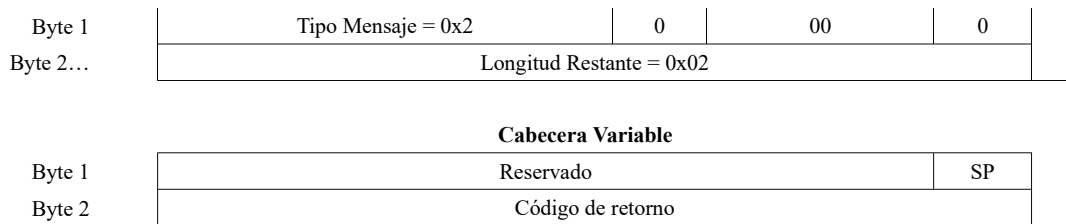


Figura 5. Formato mensaje CONNACK.

Del mensaje CONNACK mostrado en la [Figura 5](#) solo comentar que en la cabecera fija, los campos *Flag DUP* y *QoS* no tienen ningún valor válido, mientras que la cabecera variable solo tendrá dos bytes de los cuales:

- **SP:** es un *flag* utilizado para indicar al cliente si la petición de borrado de la información de la sesión, mediante el *flagClearSession* del mensaje CONNECT se ha realizado o no.
- **Código de retorno:** es utilizado para indicar al cliente el resultado del establecimiento de la conexión. Los posibles valores son mostrados en la [Tabla 1](#).

Tabla 1.- Interpretación del Código de retorno de un mensaje CONNACK.

Valor	Respuesta	Descripción
0x00	Conexión aceptada	La conexión se realizó con éxito
0x01	Conexión rechazada, versión del protocolo inaceptable	El bróker no soporta la versión del protocolo MQTT utilizada por el cliente.
0x02	Conexión rechazada, identificador rechazado	El valor del Cliente_ID tiene un formato UTF-8 correcto, pero no puede ser aceptado por el bróker.
0x03	Conexión rechazada, el servidor no está disponible.	La conexión TCP se ha establecido, pero el servicio MQTT no está disponible.
0x04	Conexión rechazada, UserName o Passwd erróneos	Los datos relativos al nombre de usuario y/o clave de usuario son erróneos.
0x04	Conexión rechazada, no autorizado	El cliente no está autorizado para conectarse a este bróker
0x06 - 0xFF		Valores reservados para uso futuro.

Por último indicar que este tipo de mensajes no tiene *Payload* asociado.

## 2.1 Ejemplo de captura.

Un ejemplo de captura de un mensaje CONNACK, realizada con la herramienta wireshark, es mostrado a continuación, donde en verde se muestra el contenido de la cabecera fija, y en azul el de la variable.

```

Frame 551: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
Ethernet II, Src: Luxshare_00:2b:c2 (3c:18:a0:00:2b:c2), Dst: Expressi_00:f1:3e (24:0a:c4:00:f1:3e)
Internet Protocol Version 4, Src: 192.168.10.8, Dst: 192.168.10.9
Transmission Control Protocol, Src Port: 1883, Dst Port: 53115, Seq: 1, Ack: 35, Len: 4
MQ Telemetry Transport Protocol
Connect Ack
0010 0000 = Header Flags: 0x20 (Connect Ack)
0010 .... = Message Type: Connect Ack (2)
.... 0... = DUP Flag: Not set
.... .00. = QoS Level: Fire and Forget (0)
.... ...0 = Retain: Not set
Msg Len: 2
.... .... 0000 0000 = Connection Ack: Connection Accepted (0)

```

### 4.3. Publicación de un tema (topic)

Una vez que se ha logrado establecer una conexión con un bróker, el cliente puede comenzar a publicar mensajes tan pronto como crea conveniente. Todo mensaje recibido por el bróker es filtrado según el *topic* al que va destinado, por ello, todos los mensajes publicados deben contener el *topic* o tema al que asocian los datos que contienen. Un nuevo *topic* o tema puede ser incluido en un mensaje cuando el cliente quiera, sin necesidad de indicárselo con antelación al bróker.

La [Figura 1](#) muestra el flujo de mensajes que se realizan durante la publicación de un tema o *topic*. Como se puede ver la publicación por parte de un cliente al bróker, o del bróker a determinados abonados de un determinado *topic* es igual, y solo requiere del establecimiento previo de la conexión.

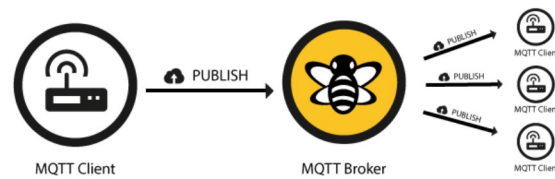


Figura 1.- Publicación de un *topic* en MQTT. (Imagen obtenida de [www.hivemq.com](http://www.hivemq.com))

## 1. Formato del mensaje PUBLISH.

Para realizar una publicación se utiliza el mensaje PUBLISH cuyo formato se muestra en la [Figura 2](#).

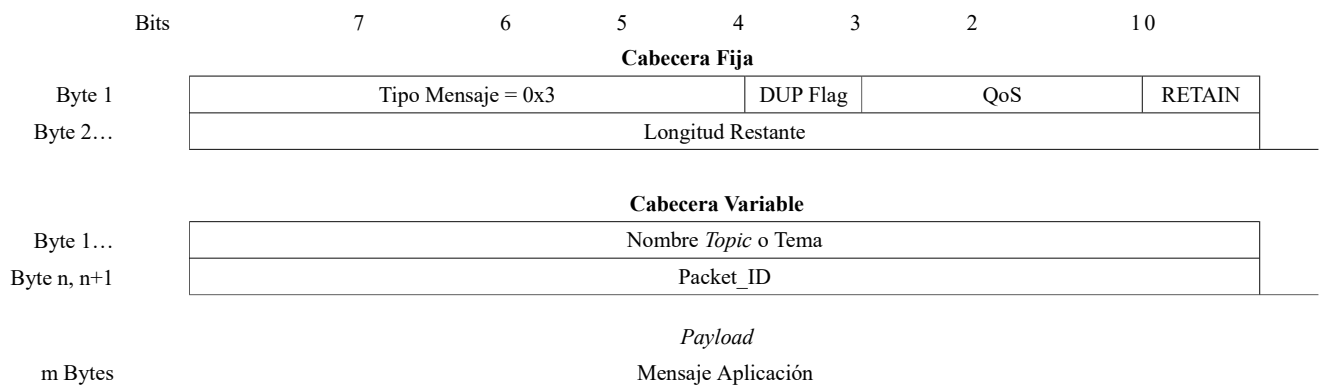


Figura 2.- Formato del mensaje PUBLISH en MQTT.

donde el contenido de la Cabecera Fija ya se describió en la sección "[Formato mensaje MQTT](#)", por lo que pasaremos a comentar el contenido de los campos de la Cabecera Variable::

- **Nombre *Topic* o Tema (número variable de bytes):** este campo indicará en formato UTF-8 cual es el identificador del *topic* o tema, según lo indicado en la sección "[Temas \(topics\) - Direccionamiento MQTT](#)", al que está asociado el mensaje. El campo tiene el formato mostrado en la [Figura 3](#):

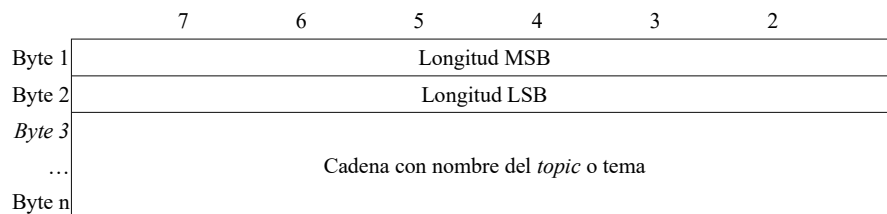


Figura 3.- Formato de los campos que indican el nombre del *topic* o tema.

donde como se puede observar los dos primeros bytes indican cual es la longitud total del campo "Nombre de *Topic*" y a partir del byte 3, el contenido es la codificación UTF-8 del nombre del *topic* según las indicaciones seguidas en la sección "[Temas \(topics\) - Direccionamiento MQTT](#)".

- **Packet\_ID (2 bytes):** son el identificador del mensaje, el cual **solo estará presente** cuando el valor del campo "QoS" de la cabecera fija sea de **1 ó 2**.
- **Mensaje Aplicación(número variable de bytes):** es el contenido del mensaje asociado al *topic* indicado en la cabecera variable. El formato de este campo es similar al del campo "Nombre *topic*": dos bytes para indicar la longitud total del campo, y a partir del tercer byte se codificará el contenido del mensaje. El contenido del mensaje se codifica en binario, ya que no se establece previamente el tipo de dato ni su formato. Por ello, la interpretación del contenido de este campo queda en manos de la aplicación final en cliente y/o bróker, que deberán saber previamente como se han codificado los datos para poder interpretarlos.

Un ejemplo de una mensaje asociado a una publicación, capturado con wireshark, se muestra a continuación. Se puede observar en verde la cabecera fija, en azul la cabecera variable, y en rojo el *payload*. Se aprecia como en este caso al tener un valor de QoS=0 no aparece el campo Packet\_ID. en la cabecera variable. Además vemos como los datos en el *payload* van en texto plano, y representa un conjunto de datos en formato JSON.

```
Frame 572: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
Ethernet II, Src: Espressi_00:f1:3e (24:0a:c4:00:f1:3e), Dst: Luxshare_00:2b:c2 (3c:18:a0:00:2b:c2)
Internet Protocol Version 4, Src: 192.168.10.9, Dst: 192.168.10.8
Transmission Control Protocol, Src Port: 53115, Dst Port: 1883, Seq: 100, Ack: 20, Len: 64
[2 Reassembled TCP Segments (66 bytes): #570(2), #572(64)]
MQ Telemetry Transport Protocol
Publish Message
0011 0000 = Header Flags: 0x30 (Publish Message)
0011 .... = Message Type: Publish Message (3)
.... 0... = DUP Flag: Not set
.... .00. = QOS Level: Fire and Forget (0)
.... ...0 = Retain: Not set
Msg Len: 64
Topic: Pycom1/measurement
Message: {"temperature":27.58,"light":28.00,"led": 0}
```



## 4.4. Suscripción a un tema (topic).

La suscripción a un *topic* o tema se puede realizar una vez que se ha establecido la conexión con el bróker. El objetivo de toda suscripción es el obtener por parte del cliente los datos asociados al tema (*topic*) al que nos suscribimos, cada vez que hay una nueva publicación de dichos datos por parte de otro cliente.

El proceso de suscripción se muestra en la [Figura 1](#). Como se puede ver el proceso consta de cuatro fases:

1. Solicitud de la suscripción por parte del cliente al bróker: Mensaje SUBSCRIBE.
2. Respuesta del éxito o no de la suscripción por parte del bróker: Mensaje SUBACK. Si la respuesta es exitosa el cliente ya está abonado al tema (*topic*) concreto luego solo tiene que esperar a que le llegue al bróker una nueva publicación sobre dicho *topic* para que éste se la reenvíe.
3. Recepción por el bróker de una publicación asociada al *topic* de la suscripción.
4. Generación de una publicación dirigida al cliente, por el bróker, con el contenido de los datos recibidos en el paso 3 por el bróker.

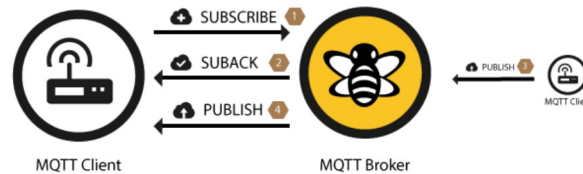


Figura 1. Suscripción a un *topic* en MQTT. (Imagen obtenida de [www.hivemq.com](http://www.hivemq.com))

Mostremos a continuación el contenido de los mensajes SUBSCRIBE y SUBACK.

## 1. Formato del mensaje SUBSCRIBE.

La [Figura 2](#) muestra el formato de un mensaje SUBSCRIBE.

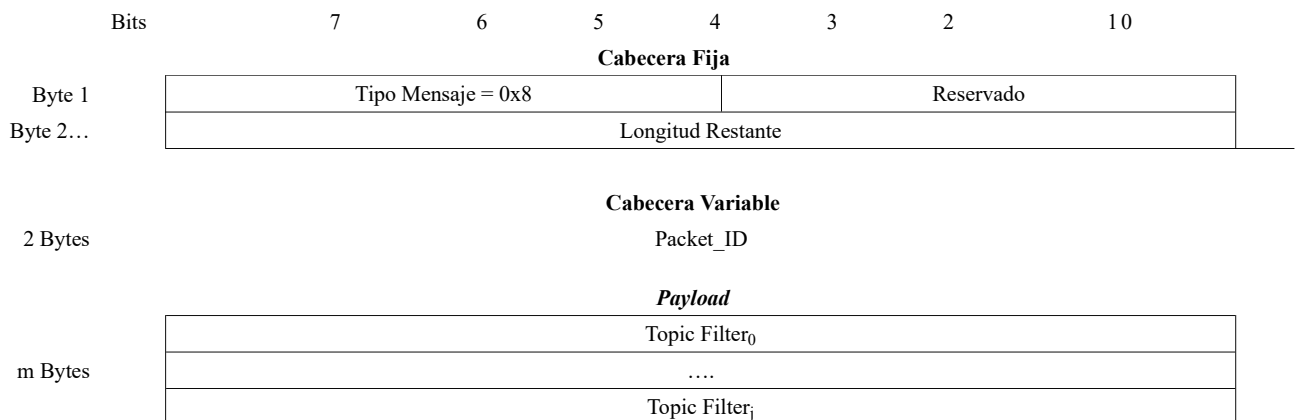


Figura 2. Formato de un mensaje SUBSCRIBE de MQTT.

En la figura anterior se puede apreciar como en la cabecera fija los campos *DUP Flag* y *QoS* no tienen ningún significado. El resto de campos se comenta a continuación:

- **Packet\_ID ( 2bytes):** especifica un identificador de mensaje, necesario para relacionar el SUBSCRIBE con el SUBACK asociado a él.
- **Topic Filter<sub>i</sub> (n bytes):** es un campo de longitud variable que indica el *topic* o tema al que el cliente se quiere suscribir, y el valor de *QoS* que quiere que se aplique a dicha suscripción. El formato de este campo se muestra en la [Figura 3](#).

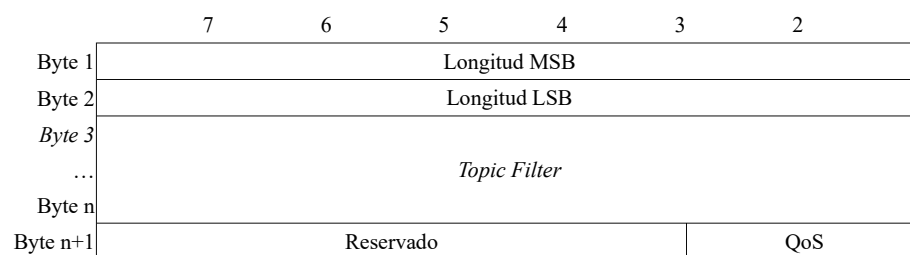


Figura 3. Formato del campo *Topic Filter* del *Payload* de un mensaje SUBSCRIBE.

Como se puede ver un campo *Topic Filter* está compuesto por:

- **Longitud MSB y Longitud LSB (2 bytes):** que indican la longitud total del campo,
- **Topic Filter (n bytes):** conjunto de bytes codificados en UTF-8 que indican un *topic* o conjunto de *topics* a los que el cliente se quiere suscribir. El formato de la cadena de caracteres que especifica el *topic* o *topics* sigue las reglas mostradas en la sección "[Temas \(topics\) - Direccionamiento MQTT](#)", teniendo en cuenta que se pueden utilizar *wildcards*.
- **QoS (2 bits):** Campo que indica el nivel de QoS: 0, 1 ó 2 que se debe aplicar al envío de los *topics* suscritos. Una descripción de dichos valores se da en la sección "[Formato mensaje MQTT](#)".

Un ejemplo de captura de un mensaje SUBSCRIBE, capturado con la herramienta wireshark, se muestra a continuación; resaltando en verde la cabecera fija, en azul la variable, y en rojo el *payload* del mensaje.

```
Frame 555: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
Ethernet II, Src: Expressi_00:f1:3e (24:0a:c4:00:f1:3e), Dst: Luxshare_00:2b:c2 (3c:18:a0:00:2b:c2)
Internet Protocol Version 4, Src: 192.168.10.9, Dst: 192.168.10.8
Transmission Control Protocol, Src Port: 53115, Dst Port: 1883, Seq: 39, Ack: 5, Len: 19
[2 Reassembled TCP Segments (23 bytes): #553(4), #555(19)]
MQ Telemetry Transport Protocol
Subscribe Request
1000 0010 = Header Flags: 0x82 (Subscribe Request)
1000 .... = Message Type: Subscribe Request (8)
.... 0... = DUP Flag: Not set
.... .01. = QOS Level: Acknowledged deliver (1)
.... ...0 = Retain: Not set
Msg Len: 21
Message Identifier: 1
Topic: Pycom1/led/state
.... ..00 = Granted Qos: Fire and Forget (0)
```

## 2. Formato mensaje SUBACK.

Cuando un bróker recibe un mensaje SUBSCRIBE de un cliente, lo procesa y le contesta con un mensaje SUBACK cuyo formato se puede ver en la [Figura 4](#).

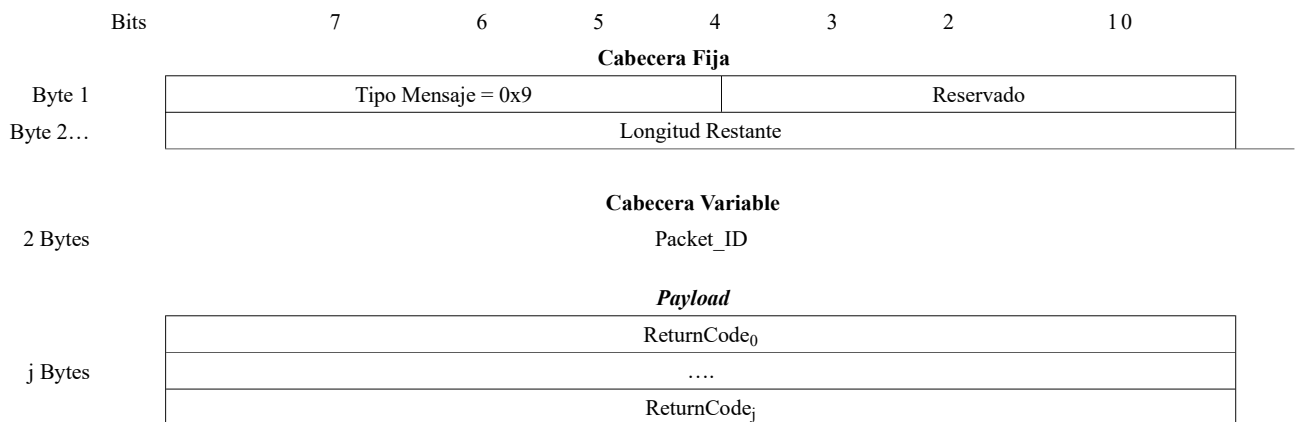


Figura 4.- Formato del mensaje SUBACK de MQTT.

La interpretación de los campos del mensaje es la siguiente:

- **Packet\_ID (2 bytes):** es el Packet\_ID del mensaje SUBSCRIBE. El campo sirve para poder relacionar en el cliente el SUBACK con su SUBSCRIBE correspondiente.
- **ReturnCode<sub>i</sub> (1 byte):** es el resultado de la suscripción al *topic* *i* incluido en el mensaje SUBSCRIBE. En un mensaje SUBACK deben existir tantos *ReturnCode* como *Topic Filter* en el mensaje SUBSCRIBE al que está asociado. La relación entre un *Topic Filter* y su *ReturnCode* viene dada por su posición en los mensajes.

Los posibles valores de este campo *returnCode* se muestran en la [Tabla 1](#).

Tabla 1.- Valores del campo *ReturnCode* de un mensaje SUBACK.

Valor	Descripción
0x00	Éxito de suscripción. QoS a aplicar 0
0x01	Éxito de suscripción. QoS a aplicar 1
0x02	Éxito de suscripción. QoS a aplicar 2
0x80	Falló la suscripción

Un ejemplo de captura de un mensaje SUBACK se muestra a continuación. En verde la cabecera fija, en azul la variable y en rojo su *payload*.

```
Frame 557: 59 bytes on wire (472 bits), 59 bytes captured (472 bits) on interface 0
Ethernet II, Src: Luxshare_00:2b:c2 (3c:18:a0:00:2b:c2), Dst: Expressi_00:f1:3e (24:0a:c4:00:f1:3e)
Internet Protocol Version 4, Src: 192.168.10.8, Dst: 192.168.10.9
Transmission Control Protocol, Src Port: 1883, Dst Port: 53115, Seq: 5, Ack: 58, Len: 5
MQ Telemetry Transport Protocol
Subscribe Ack
1001 0000 = Header Flags: 0x90 (Subscribe Ack)
1001 .... = Message Type: Subscribe Ack (9)
.... 0... = DUP Flag: Not set
.... .00. = QOS Level: Fire and Forget (0)
.... ...0 = Retain: Not set
Msg Len: 3
Message Identifier: 1
.... ..00 = Granted Qos: Fire and Forget (0)
```

## 4.5. QoS en MQTT

En MQTT el nivel de QoS (*Quality of Service*) es un acuerdo entre el cliente y el broker destinado a garantizar la entrega de los mensajes (publicaciones). En MQTT existen tres niveles de QoS:

- Nivel 0 - "*At most once*".
- Nivel 1 - "*At least once*".
- Nivel 2 - "*Exactly once*".

Cada uno de estos niveles determinará la necesidad de envío de confirmaciones tras la recepción de un mensaje PUBLISH, el mecanismo de confirmación y la existencia o no de retransmisiones. A continuación se describe detenidamente cada uno de estos mecanismos.

### 1. QoS 0 - "*At most once*".

El nivel 0 es el mínimo nivel de QoS, y se relaciona con un servicio "*best-effort*". El mensaje PUBLISH enviado no necesita ser confirmado por el receptor, por lo que MQTT no se encarga de asegurar su entrega. Los mecanismos de control de error se dejan a el protocolo TCP. Este nivel de servicio también se suele llamar "*Fire and Forget*" haciendo hincapié en la idea de que el mensaje es enviado y no reparamos más en él. La [Figura 1](#) representa los mensajes relacionados con la publicación de un *topic* en este nivel de QoS.



Figura 1.- Mensajes utilizados para la publicación de un *topic* con un nivel QoS = 0. (Figura obtenida de [www.hivemq.com](http://www.hivemq.com)).

Como se observa en la [Figura 1](#), en este nivel de QoS = 0 solo se envía un mensaje PUBLISH (su descripción está en la sección "[Publicación de un tema \(topic\)](#)") para la publicación de un *topic*. No necesitando ninguna confirmación por parte del receptor.

### 2. QoS 1 - "*At least once*".

En el nivel 1 de QoS, se garantiza la entrega del mensaje al destinatario al menos una vez. Pero el mensaje puede ser recibido más de una vez por el destinatario, por lo que se deberán de gestionar la entrega de **mensajes duplicados**. Para ello se emplea el *flag* "**DUP flag**" de la cabecera fija del mensaje PUBLISH, el cual se pondrá a 1 cuando el mensaje sea reenviado. El proceso de publicación en este nivel QoS = 1 se muestra en la [Figura 2](#).

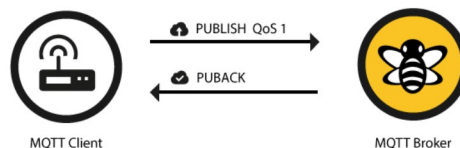


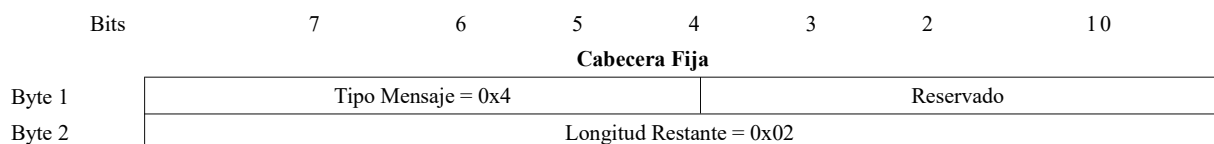
Figura 2.- Mensajes involucrados en la publicación de un *topic* para un nivel QoS=1. (Figura obtenida de [www.hivemq.com](http://www.hivemq.com))

Como se observa, el receptor de un mensaje PUBLISH con el flag de QoS=1, debe responder a la recepción del mismo con un mensaje PUBACK. Si este último mensaje no es recibido por el cliente, éste deberá de volver a reenviar el mensaje PUBLISH, ahora con el campo "*DUP flag*" a 1, para indicar que el mensaje es duplicado de un mensaje anterior. Y de esta forma alertar al receptor de posibles duplicados.

El formato del mensaje PUBACK se muestra a continuación.

#### 2.1. Formato mensaje PUBACK.

Este mensaje se envía siempre que se recibe un mensaje PUBLISH con el campo QoS = 0x01. Su contenido se representa en la [Figura 3](#).



**Cabecera Variable**

2 Bytes

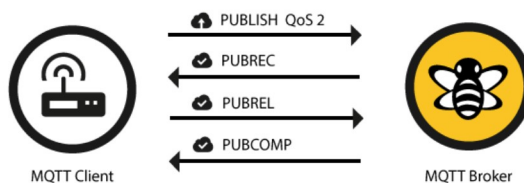
Packet\_ID

Figura 3.- Formato del mensaje PUBACK de MQTT.

Como se aprecia el mensaje PUBACK solo tiene un campo de cabecera Fija, donde se indica el tipo de mensaje, valor 0x04 en referencia a PUBACK, y en una Cabecera Variable, donde mediante dos bytes se indica el **Packet\_ID**, que es el identificador del paquete PUBLISH enviado previamente y con el que está relacionado este PUBACK. De este modo la relación entre un mensaje PUBLISH y otro PUBACK se establece siempre que ambos tengan el mismo valor de Packet\_ID.

### 3. QoS 2 - "*Exactly once*".

En nivel QoS=2 es el mayor nivel de calidad de servicio que puede proporcionar MQTT. En este nivel se asegura que un mensaje PUBLISH es entregado "una y solo una vez", evitándose la aparición de duplicados en recepción. El proceso de publicación de un *topic* utilizando el nivel QoS 2 se muestra en la [Figura 4](#).

Figura 4.- Mensajes involucrados en la publicación de un *topic* para un nivel QoS=2. (Figura obtenida de [www.hivemq.com](http://www.hivemq.com))

El proceso de publicación en este nivel de QoS sigue los siguientes pasos.

1. El emisor envía un mensaje PUBLISH con el campo QoS=2. Se guarda una copia del mensaje en memoria por si tuviera que reenviarlo.
2. Cuando el receptor recibe el mensaje PUBLISH con el campo QoS=2 hace lo siguiente:
  - Procesa el mensaje, y guarda una copia en memoria, hasta que recibe un mensaje PUBREL. Así se evita el tener que procesar el mensaje recibido una segunda vez.
  - Envía un mensaje PUBREC al emisor.
3. A la recepción del mensaje PUBREC el emisor del PUBLISH hace lo siguiente:
  - Elimina de memoria la copia del mensaje PUBLISH, ya que se le ha confirmado su recepción.
  - Guarda una copia del mensaje PUBREC.
  - Envía un mensajr PUBREL.
4. Cuando el receptor recibe el mensaje PUBREL hace lo siguiente:
  - Libera la memoria con la copia del mensaje PUBLISH que tenía almacenado.
  - Envía un mensaje PUBCOMP al emisor.
5. El emisor recibe el mensaje PUBCOMP:
  - Libera toda la memoria asociada a la copia del mensaje PUBREC.
  - Da por finalizada la entrega con éxito.

En el caso de que se produzca alguna pérdida de algún mensaje, la política de reenvíos es la siguiente:

- **Reenvío del mensaje PUBLISH:** Solo se produce cuando el emisor del mensaje PUBLISH no ha recibido un PUBREC. Una vez recibido dicho mensaje no se podrá reenviar el mensaje PUBLISH, puesto que ya se da por sentado que ha sido recepcionado correctamente por el destinatario.
- **Reenvío del mensaje PUBREC:** Cuando no se recibe un mensaje PUBREL.
- **Reenvío del mensaje PUBREL:** Cuando no se recibe un mensaje PUBCOMP.
- **Reenvío del mensaje PUBCOMP:** Se producirá cuando se reciba la recepción de un PUBREL repetido, debido a la retransmisión del mismo por el emisor al no haber recibido el PUBCOMP anterior.

### 4. ¿Qué nivel de QoS usar?

Dado que el protocolo MQTT en el caso de la IoT se suele utilizar en redes de poco ancho de banda, y con dispositivos editores/abonados con autonomía energética limitada, se ha de procurar elegir el nivel QoS más adecuado a cada circunstancia, ya que cuanto mayor es el nivel mayor es el número de mensajes a transmitir y por lo tanto mayor es el consumo de ancho de banda, y también aumenta el consumo energético asociado a la transmisión de mensajes. A continuación se dan unas pautas de cuando utilizar un nivel de QoS u otro.

#### Usaremos QoS = 0 cuando...

- Siempre que exista una conexión robusta entre emisor y receptor. Caso conexiones cableadas.

- No nos importe la pérdida de algún mensaje publicado. Por ejemplo, cuando la pérdida de un mensaje no es crítica, bien por que su información no es muy importante, o porque la frecuencia de envío del mismo puede suplir una pérdida puntual del mismo.
- No necesitamos un encolado de los mensajes, para su posterior envío a abonados una vez estos se suscriban al *topic* encolado.

### Usaremos QoS = 1 cuando ...

- Queremos asegurarnos de no perder ningún mensaje publicado, pero no nos importa recibir mensajes duplicados.
- Queremos asegurarnos de que un nuevo abonado reciba copia del último mensaje asociado al *topic* recibido antes de la suscripción del abonado.
- Deseamos tener lo anterior pero sin mucho sobreflujo de control.

### Usaremos QoS =2 cuando...

- No queremos perder ningún dato, pero además no queremos tener duplicados de mensajes.
- Queremos asegurarnos de que un nuevo abonado a un *topic* no tenga que esperar a una nueva publicación relativa a dicho *topic* para recibir información sobre él. Sino que en el momento de la suscripción le queremos enviar el estado del *topic* según la última publicación.

## 5. Bróker MQTT

Para la implementación de un bróker de MQTT es necesaria la instalación de algún servidor del protocolo MQTT al que se puedan conectar los clientes, bien publicando o bien abonándose a algún *topic*.

Hay una gran variedad de servidores del protocolo MQTT entre ellos podemos destacar:

- [moquette](#)
- [Mosquitto](#)
- [MQTTRoute](#)

En nuestro caso nos centraremos en el bróker [mosquitto](#).

## 5.1. Bróker Eclipse Mosquitto

Eclipse Mosquitto™ es un proyecto de código abierto de [iot.eclipse.org](https://iot.eclipse.org) que implementa un bróker para el protocolo MQTT tanto en su versión v3.1 como en la v3.1.1.

### Instalación

La instalación del Eclipse Mosquitto™ dependerá del sistema operativo en el que se quiera instalar. Una guía detallada de los pasos a seguir en cada una de las arquitecturas se puede obtener en <https://mosquitto.org/download/>. Nosotros nos centraremos en los pasos a seguir para un sistema Linux con una distribución Debian

### Instalación en Linux Ubuntu/Debian

Tanto para las distribuciones Ubuntu como Debian, en sus últimas versiones, el bróker mosquitto ya está incluido en sus repositorios respectivo. Aunque puede la versión incluida no sea la última del bróker, y haya que esperar unos meses para que sea incorporada. Aun así, y a no ser que se quiera tener siempre la última versión, se recomienda el utilizar la versión incluida en el repositorio de la distribución, tanto por facilidad de instalación, como por estabilidad del código.

La instalación es muy sencilla. Tan solo habrá que ejecutar los siguientes comandos:

```
$ sudo apt-get update
```

```
$ sudo apt-get install mosquitto
```

El primero de ellos actualizará la lista de paquetes incluidas en los repositorios de la distribución, y la última instalará el bróker.

Una vez finalizada la distribución, el bróker es configurado para arrancarse ineditamente, y al inicializar la máquina. Por lo que el usuario no deberá de hacer nada más para tener el bróker mosquitto activo.



## 6. Cliente MQTT

Como ya se ha comentado anteriormente, todo editor o abonado deberá implementar un cliente MQTT para poder comunicarse con el servidor o bróker MQTT. Existen diversas implementaciones de MQTT, pero nosotros nos centraremos en describir el uso de las dos implementaciones que utilizaremos en la asignatura:

- [Cliente MQTT para Micro-Python de Pycom®](#).
- [Cliente MQTT en Node-Red](#).

Ambos serán descritos en sucesivas secciones.

## 6.1. Cliente MQTT para Micro-Python.

En esta sección vamos a comentar el uso de la librería `mqtt.py` desarrollada por Pycom® para la implementación de un cliente MQTT en sus dispositivos a partir del lenguaje de programación Micro-Python. Dicha librería está accesible en el siguiente repositorio de GitHub <https://github.com/pycom/pycom-libraries.git> dentro del directorio `lib/mqtt`.

### 1. La clase MQTTClient.

La librería define una clase denominada `MQTTClient` que implementa un cliente MQTT, y que se describe a continuación.

#### Constructor

```
class mqtt.MQTTClient(client_id, server, port=0, user=None, password=None, keepalive=0, ssl=False,
ssl_params={})
```

Crea un nuevo cliente MQTT con los siguientes parámetros:

- **client\_id (string)**: identificador del cliente, debe ser único e identifica al cliente frente al bróker.
- **server (string)**: es una cadena de caracteres que identifica al bróker con el que me conectaré. Puede ser un nombre de dominio DNS o una dirección IPv4.
- **port (integer)**: entero que indica el puerto en el que el bróker está a la escucha. En caso de no especificarlo se utilizará el puerto TCP 1883. Y si utilizamos SSL (parámetro `ssl=True`) se asignará el 8883.
- **user (string)**: nombre del usuario con el que nos autenticaremos en el bróker.
- **password (string)**: clave del usuario para autenticarnos en el bróker.
- **keepalive (integer)**: intervalo de tiempo máximo entre dos envíos por parte del cliente. Si no se especifica nada el tiempo se considera infinito.
- **ssl (True/False)**: Indica la utilización de SSL para la encriptación de los datos transmitidos.
- **ssl\_params (dict)**: Diccionario en python con los parámetros a usar en la configuración de la conexión SSL.

#### Métodos

- **mqtt.connect(clean\_session=True)**

Establece una conexión con el bróker, enviándole un mensaje `CONNECT`. `clean_session` indica si la sesión debe ser inicializada tras la conexión o retomamos el estado de la conexión anterior en caso de haber existido.

Por lo tanto el valor `clean_session` indicado será el valor del *flag Clean Session* del mensaje `CONNECT`.

- **mqtt.disconnect()**

Realiza la desconexión del cliente con el bróker, enviándole un mensaje `DISCONNECT`.

- **mqtt.ping()**

Envía un mensaje `PINGREQ` al bróker

- **mqtt.publish(topic, msg, retain=False, qos=0)**

Realiza la publicación de un *topic* o tema. Los parámetros son:

- **topic (string)**: cadena en formato UTF-8 según lo indicado en la sección "[Temas \(topics\) - Direccionamiento MQTT](#)" que indica el *topic* con el que se relaciona el mensaje enviado.
- **msg (array de bytes)**: un array de bytes que contiene los datos a enviar. La interpretación y formato de estos datos es libre y dependerá de la aplicación en sí. Es dicha aplicación la que debe dar formato a los datos e interpretarlos
- **retain (True/False)**: valor a asignar al *flag* `RETAIN` del mensaje `PUBLISH`. Por lo tanto, este valor determinará si los datos contenidos en `msg` deben ser guardados en memoria por el bróker, para su envío inmediato a un cliente que se suscriba a dicho *topic*, sin esperar a nuevas publicaciones.
- **qos (integer)**: valor del nivel de QoS a aplicar a esta publicación. Como se comentó en la sección "[QoS en MQTT](#)" sus valores pueden ser 0, 1 ó 2.

- **mqtt.subscribe(self, topic, qos=0)**

Realiza una suscripción al *topic* indicado en el campo `topic`. El significado de cada parámetro es similar al del método `mqtt.publish()`.

- **mqtt.wait()**

Espera hasta la recepción de un mensaje MQTT y lo procesa. El procesamiento del mensaje se realizará por la función de *callback* definida con `mqtt.set_callback()`.

- **`mqtt.check_msg()`**

Testea si hay un mensaje MQTT recibido a la espera de ser procesado. Si no lo hubiera retorna de inmediato. En caso de existir el mensaje realiza la misma acción que `mqtt.wait()`.

- **`mqtt.set_callback(callback_function)`**

Determina la función *function* a la que llamar para el procesamiento de cualquier mensaje recibido procedente del bróker.

¿Cómo debe ser el prototipo de la `callback_function()`?

La función de *callback* debe tener el siguiente prototipo:

```
def callback_funcion (topic, msg)
```

donde:


- **`topic (string)`**: es el *topic* con el que está relacionado el mensaje recibido, y que deberá coincidir con uno de los *topic* a los que el cliente se ha suscrito previamente.
- **`msg (array de bytes)`**: son los datos enviados en el mensaje recibido. Su formato debe ser conocido previamente para poder interpretarlos.

Un ejemplo de uso se puede encontrar en <https://docs.pycom.io/chapter/tutorials/all/mqtt.html>

## 6.2. Cliente MQTT en Node-Red

**Node-Red** tiene incorporado un cliente de MQTT, por lo que en esta sección nos centraremos únicamente en los pasos que habrá que realizar para realizar una publicación o una suscripción a un *topic*.

### Publicación de un *topic*.

Para la publicación de un *topic* o tema en Node-Red deberemos de seleccionar del menú de **nodos output** (columna izquierda de la interfaz de usuario) el nodo **mqtt** . Una vez situado en el espacio de trabajo, pulsaremos sobre él y se obtendrá el menú de configuración como se muestra en la [Figura 1](#).

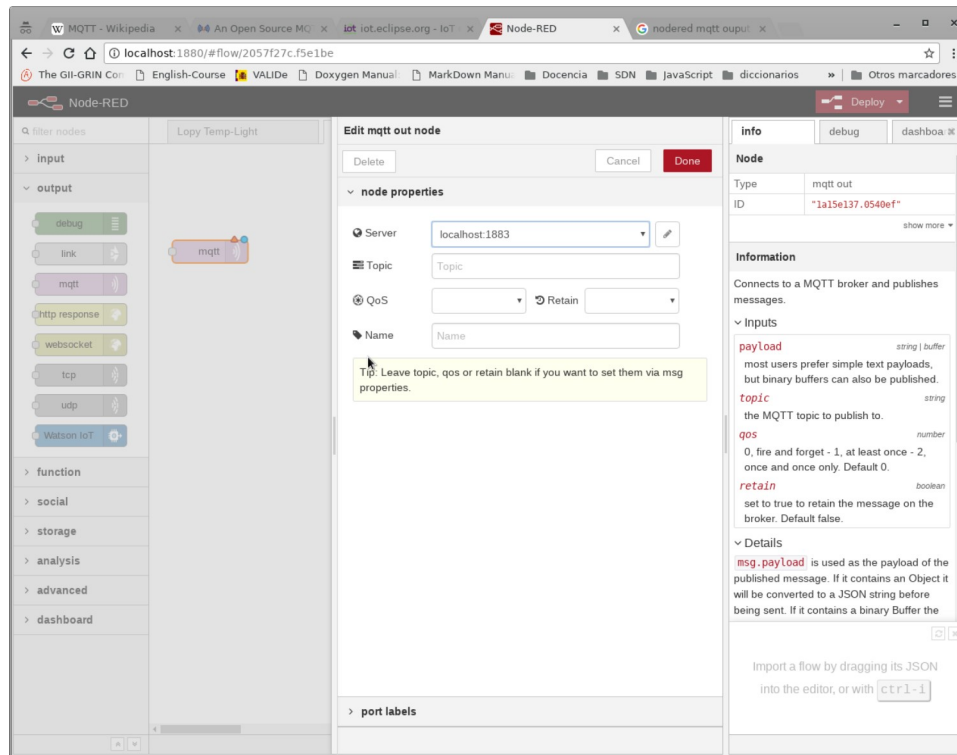




Figura 1.- Menú de configuración del nodo de publicación MQTT en Node-Red.

Como se puede observar los parámetros de configuración del nodo MQTT para una publicación serán:

- **"Server"**: Hay que especificar los distintos parámetros de configuración de la conexión, pulsando sobre el botón . Entre los parámetros a configurar están:
  - Pestaña **"Connettion"**: especifica los parámetros relativos a la IP o nombre de dominio del servidor, puerto en el que está a la escucha (1883 por defecto), uso de SSL, ID del cliente en caso, tiempo entre mensajes "Keep alive time", borrado del histórico cada vez que se realiza una nueva conexión "Use clean session".
  - Pestaña **"Security"**: parámetros relativos a la autenticación y autorización, **"Username"** y **"Password"**.
  - Pestaña **"Birth Message"** (opcional): mensaje inicial. Se especifica el *topic* **"Topic"**, la **"QoS"** del mensaje, si este se guardará o no en el bróker **"Retain"** y el mensaje a enviar **"Payload"**.
  - Pestaña **"Will Message"** (opcional): mensaje que se enviará al bróker cuando la conexión se cierre. Se especifica el *topic* **"Topic"**, la **"QoS"** del mensaje, si este se guardará o no en el bróker **"Retain"** y el mensaje a enviar **"Payload"**.
- **"Topic"**: se indica el *topic* o tema con el que se relaciona este nodo.
- **"QoS"**: nivel de QoS que se utilizará para tratar a los mensajes de este tema o *topic*.
- **"Retain"**: si el bróker guardará el último mensaje recibido para su comunicación cada vez que un nuevo cliente se suscriba a este *topic*.
- **"Name"**: Nombre que aparecerá en el icono del nodo en el área de trabajo de la interfaz de node-red.

### Suscripción a un *topic*.

Para la suscripción a un *topic* o tema en Node-Red deberemos de seleccionar del menú de **nodos input** (columna izquierda de la interfaz de usuario) el nodo **mqtt** . Una vez situado en el espacio de trabajo, pulsaremos sobre él y se obtendrá el menú de configuración como se muestra en la [Figura 2](#).

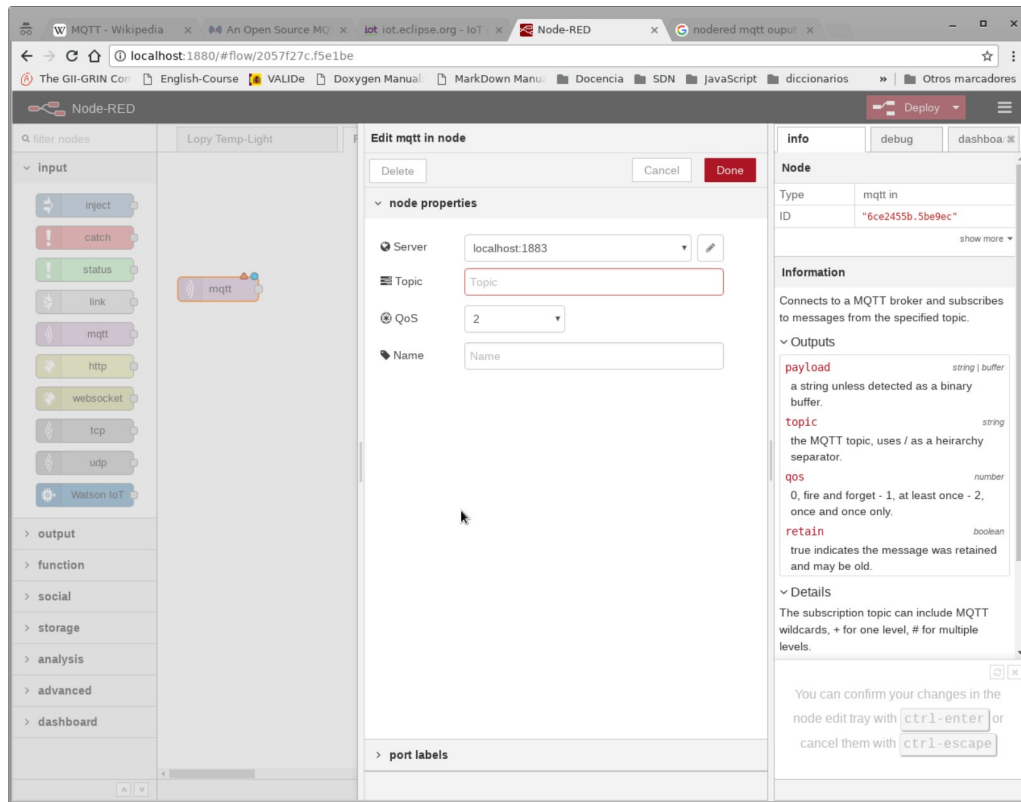


Figura 2.- Menú de configuración del nodo de suscripción MQTT en Node-Red.

Los **parámetros de configuración** para el caso del nodo de suscripción de MQTT en node-red son **similares al caso de la publicación** (ver caso anterior), solo que ahora el campo "Retain" no tiene sentido al ser una suscripción.

## 7. Bibliografía

- HiveMQ Enterprise MQTT Broker. "**MQTT Essentials**". <https://www.hivemq.com/mqtt-essentials/>. Accedido el 17/01/2018.
- Steve's internet Guide. "**MQTT Protocol Overview -Beginners Guide**". <http://www.steves-internet-guide.com/mqtt/>. Accedido el 17/01/2018.
- Eutothech,IBM<sup>®</sup>. "**MQTT V3.1 Protocol Specification**". [http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT\\_V3.1\\_Protocol\\_Specific.pdf](http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf). Accedido el 17/01/2018.
- Andrew Banks and Rahul Gupta. "**MQTT Version 3.1.1 Plus Errata 01**". 10 December 2015. OASIS Standard Incorporating Approved Errata 01. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>. Accedido el 17/01/2018
- Ken Borgendale and Brian Raymor. "**Big Ideas for MQTT v5**". <https://www.oasis-open.org/committees/download.php/57616/Big%20Ideas%20for%20MQTT%20v5.pdf>. Accedido el 17/01/2018
- Pycom<sup>®</sup>. "MQTT library for Pycom devices". <https://github.com/pycom/pycom-libraries/blob/master/lib/mqtt/mqtt.py>. Accedido 17/01/2018.
- Eclipse Foundation. "Eclipse Mosquitto<sup>™</sup>". <https://mosquitto.org/>. Accedido 24/01/2018