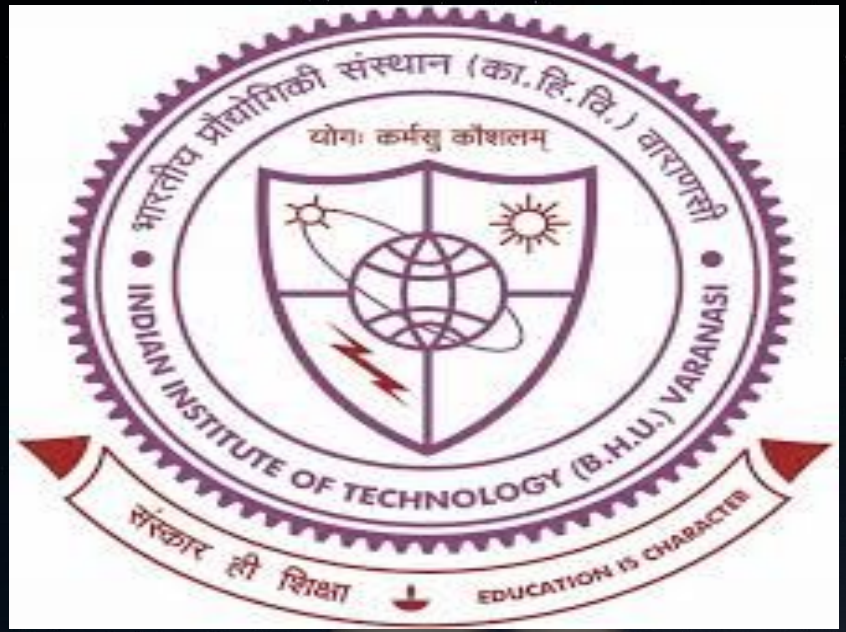
Message Queuing Telemetry Transport (**MQTT**)



By : Sidhant Sarraf(17074015)
Under Guidance of Dr. Hari Prabhat Gupta

MQTT

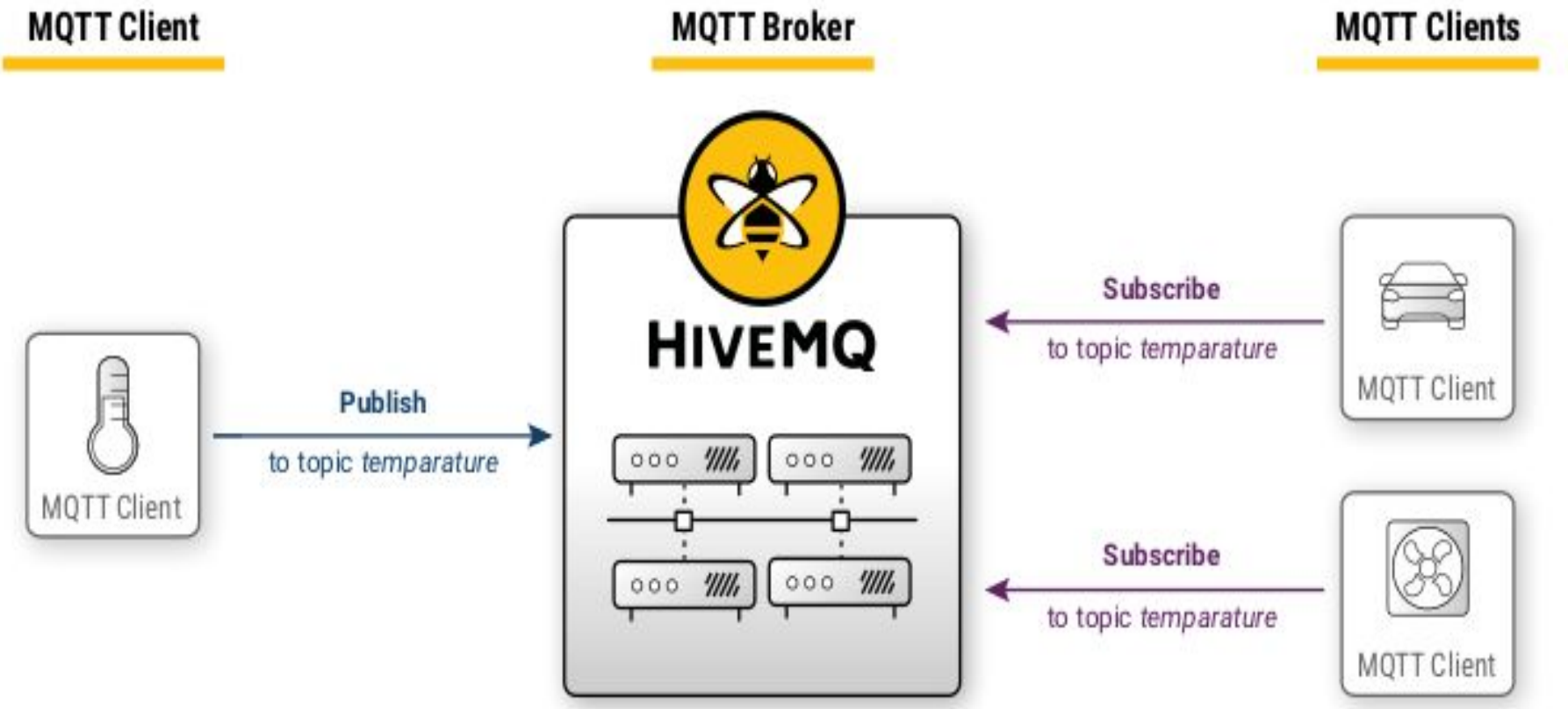
“MQTT is a Client Server publish/subscribe messaging transport protocol. It is lightweight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.”

The Publish Subscribe Pattern

The pub/sub model removes direct communication between the publisher of the message and the recipient/subscriber. The filtering activity of the broker makes it possible to control which client/ subscriber receives which message. The decoupling has three dimensions: space, time, and synchronization:

- **Space decoupling:** Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).
- **Time decoupling**: Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling:** Operations on both components do not need to be interrupted during publishing or receiving.

MQTT Architecture



Distinction: MQTT & Message Queues

- **In a message queue, each incoming message is stored in the queue until it is picked up by a client** (often called a consumer). If no client picks up the message, the message remains stuck in the queue and waits to be consumed. In a message queue, it is not possible for a message not to be processed by any client, as it is in MQTT if nobody subscribes to a topic.

- **In a traditional message queue, a message can be processed by one consumer only.** The load is distributed between all consumers for a queue. In MQTT the behavior is quite the opposite: every subscriber that subscribes to the topic gets the message.

- **Queues are named and must be created explicitly.** A queue is far more rigid than a topic. Before a queue can be used, the queue must be created explicitly with a separate command. Only after the queue is named and created is it possible to publish or consume messages. In contrast, MQTT topics are extremely flexible and can be created on the fly.

Client, Broker and Connection Establishment

- ## Client

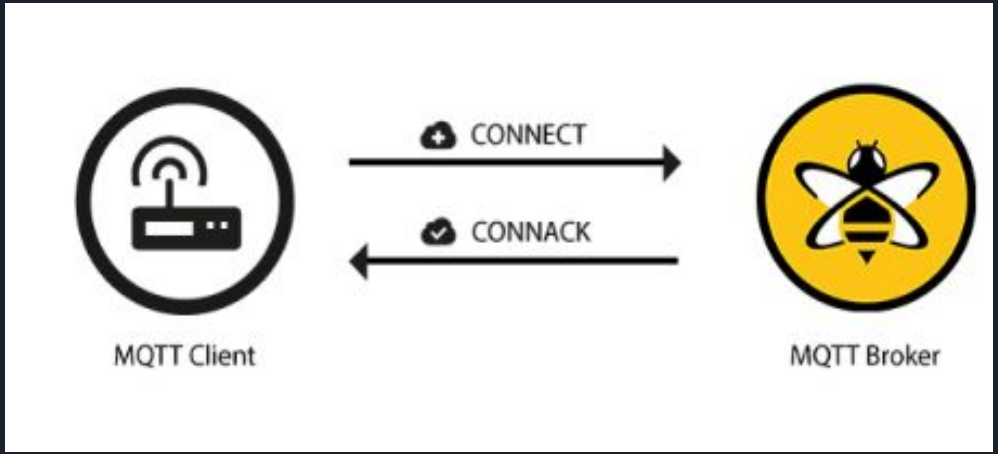
MQTT clients are both publishers and subscribers. The publisher and subscriber labels refer to whether the client is currently publishing messages or subscribing to messages (publish and subscribe functionality can also be implemented in the same MQTT client).

- ## Broker

The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients. It also holds the sessions of all persisted clients, including subscriptions and missed messages. Another responsibility of the broker is to authenticate and authorize the clients.

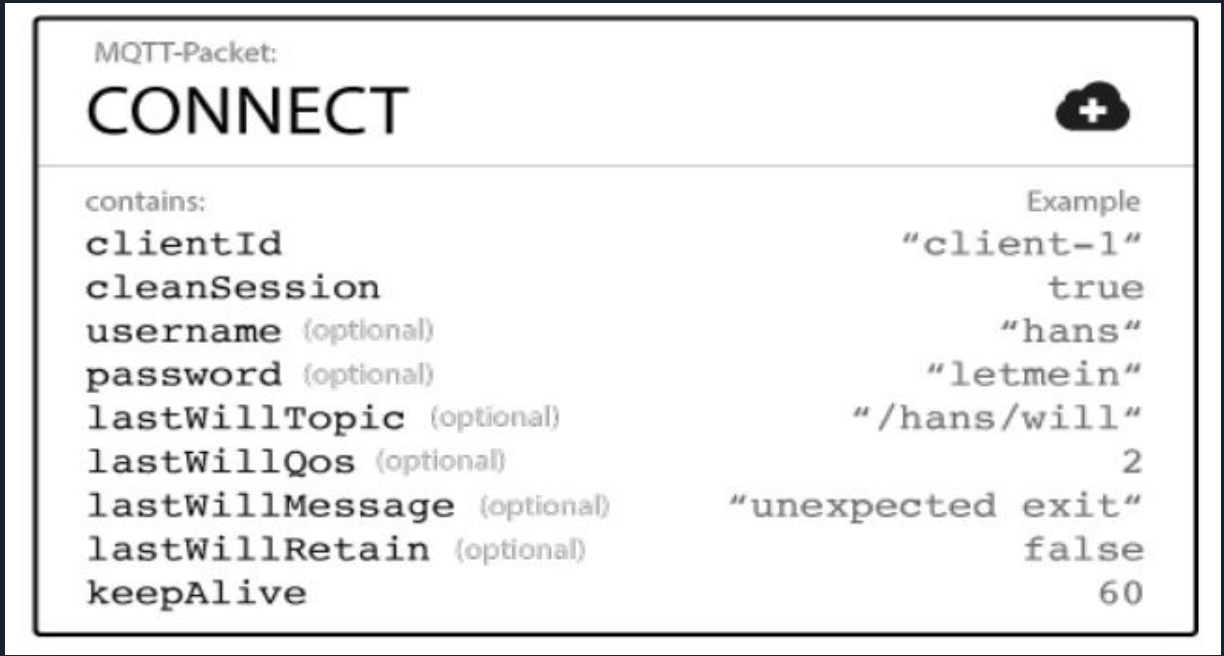
MQTT Connection

- The MQTT protocol is based on TCP/IP. Both the client and the broker need to have a TCP/IP stack.
- To initiate a connection, **the client sends a CONNECT message to the broker. The broker responds with a CONNACK (Acknowledgement)** message and a status code. Once the connection is established, the broker keeps it open until the client sends a disconnect command or the connection breaks.



CONNECT Message - Client Initiates Connection

To initiate a connection, the client sends a command message to the broker. If this CONNECT message is malformed (according to the MQTT specification) or too much time passes between opening a network socket and sending the connect message, the broker closes the connection.



Connect Message Cont.

ClientId: The client identifier (ClientId) identifies each MQTT client that connects to an MQTT broker. The broker uses the ClientID to identify the client and the current state of the client.Therefore, this ID should be unique per client and broker.

Clean Session: The clean session flag tells the broker whether the client wants to establish a persistent session or not. In a persistent session (CleanSession = false), the broker stores all subscriptions for the client and all missed messages for the client that subscribed with a Quality of Service (QoS) level 1 or 2. If the session is not persistent (CleanSession = true), the broker does not store anything for the client and purges all information from any previous persistent session.

Connect Message Cont.

Will Message: The last will message is part of the Last Will and Testament (LWT) feature of MQTT. This message notifies other clients when a client disconnects ungracefully.

KeepAlive: The keep alive is a time interval in seconds that the client specifies and communicates to the broker when the connection established. This interval defines the longest period of time that the broker and client can endure without sending a message

Username/Password: MQTT can send a user name and password for client authentication and authorization. However, if this information isn't encrypted or hashed (either by implementation or TLS), the password is sent in plain text.

CONNACK Message - Broker Response

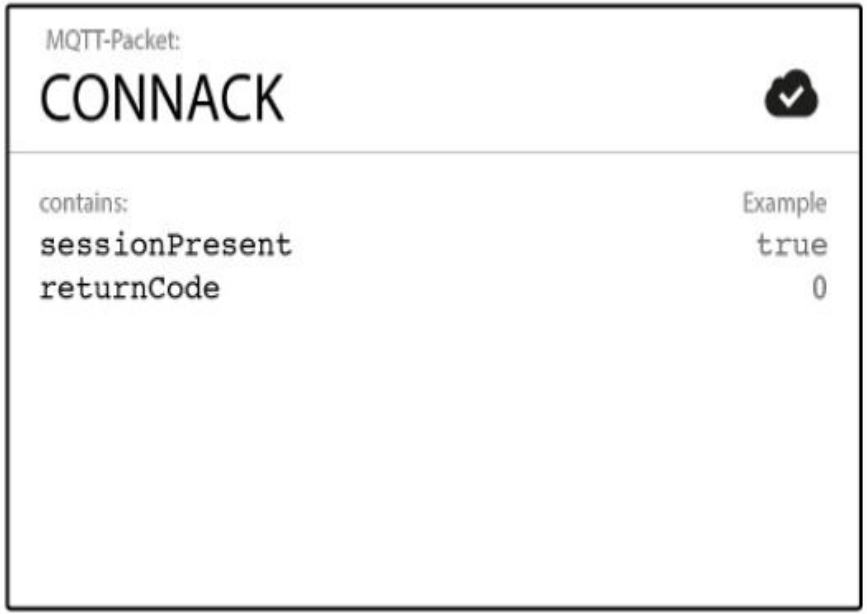
Session Present Flag:

The session present flag tells the client whether the broker already has a persistent session available from previous interactions with the client. When a client connects with Clean Session set to true, the session present flag is always false because there is no session available. If a client connects with Clean Session set to false, there are two possibilities: If session information is available for the client Id and the broker has stored session information, the session present flag is true. Otherwise, if the broker does not have any session information for the client ID, the session present flag is false.

Connect Acknowledge Flag:

The second flag in the CONNACK message is the connect acknowledge flag. This flag contains a return code that tells the client whether the connection attempt was successful.

CONNACK Message - Broker Response



Return Code	Return Code Response
0	Connection accepted
1	Connection refused, unacceptable protocol version
2	Connection refused, identifier rejected
3	Connection refused, server unavailable
4	Connection refused, bad user name or password
5	Connection refused, not authorized

Publish

Each message must contain a topic that the broker can use to forward the message to interested clients. Typically, each message has a payload which contains the data to transmit in byte format.

A PUBLISH message in MQTT has several attributes that we want to discuss in detail:

Topic Name: The topic name is a simple string that is hierarchically structured with forward slashes as delimiters.

QoS: This number indicates the Quality of Service Level (QoS) of the message. There are three levels: 0,1, and 2. The service level determines what kind of guarantee a message has for reaching the intended recipient (client or broker).

Payload: This is the actual content of the message. MQTT is data-agnostic. It is possible to send images, text in any encoding, encrypted data, and virtually every data in binary.

Retain Flag: This flag defines whether the message is saved by the broker as the last known good value for a specified topic. When a new client subscribes to a topic, they receive the last message that is retained on that topic.

Packet Identifier: The packet identifier uniquely identifies a message as it flows between the client andbroker.

Publish

MQTT-Packet:

PUBLISH

contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

Subscribe

Publishing a message doesn't make sense if no one ever receives it. In other words, if there are no clients to subscribe to the topics of the messages.

Packet Identifier: The packet identifier uniquely identifies a message as it flows between the client and broker. The client library and/or the broker is responsible for setting this internal MQTT identifier.

List of Subscriptions: A SUBSCRIBE message can contain multiple subscriptions for a client. Each subscription is made up of a topic and a QoS level. The topic in the subscribe message can contain wildcards that make it possible to subscribe to a topic pattern rather than a specific topic.

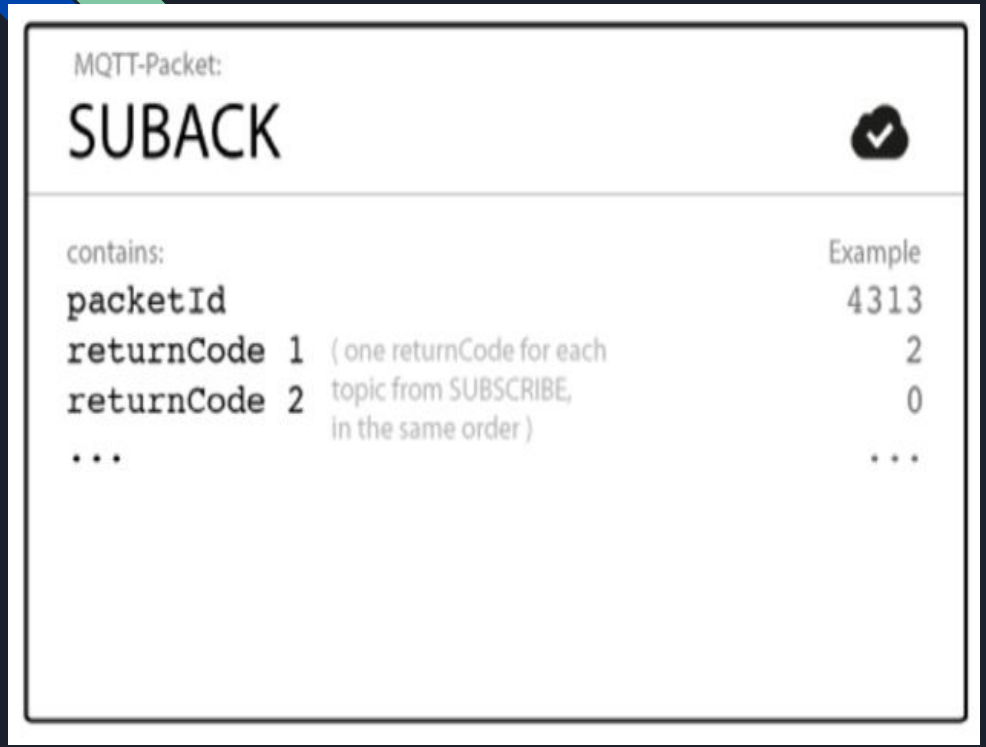
Suback

To confirm each subscription, the broker sends a SUBACK acknowledgement message to the client. This message contains the packet identifier of the original Subscribe message (to clearly identify the message) and a list of return codes.

Packet Identifier: The packet identifier is a unique identifier used to identify a message. It is the same as in the SUBSCRIBE message.

Return Code: The broker sends one return code for each topic/QoS-pair that it receives in the SUBSCRIBE message. For example, if the SUBSCRIBE message has five subscriptions, the SUBACK message contains five return codes. The return code acknowledges each topic and shows the QoS level that is granted by the broker. If the broker refuses a subscription, the SUBACK message conains a failure return code for that specific topic. For example, if the client has insufficient permission to subscribe to the topic or the topic is malformed.

Suback



Return Code	Return Code Response
0	Success - Maximum QoS 0
1	Success - Maximum QoS 1
2	Success - Maximum QoS 2
128	Failure

Unsubscribe

The counterpart of the SUBSCRIBE message is the UNSUBSCRIBE message. This message deletes existing subscriptions of a client on the broker. The UNSUBSCRIBE message is similar to the SUBSCRIBE message and has a packet identifier and a list of topics.

Packet Identifier: The packet identifier uniquely identifies a message as it flows between the client and broker. The client library and/or the broker is responsible for setting this internal MQTT identifier.

List of Topic: The list of topics can contain multiple topics from which the client wants to unsubscribe. It is only necessary to send the topic (without QoS). The broker unsubscribes the topic, regardless of the QoS level with which it was originally subscribed.

Topics & Best Practices

In MQTT, the word topic refers to an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

Wildcards:

When a client subscribes to a topic, it can subscribe to the exact topic of a published message or it can use wildcards to subscribe to multiple topics simultaneously. A wildcard canonly be used to subscribe to topics, not to publish a message. There are two different kinds of wildcards: single-level and multi-level.

- Single Level: +
- Multilevel: #

Quality of Service Levels

The Quality of Service (QoS) level is an agreement between the sender of a message and the receiver of a message that defines the guarantee of delivery for a specific message. There are 3 QoS levels in MQTT:

- At most once (0)
- At least once (1)
- Exactly once (2).

When you talk about QoS in MQTT, you need to consider the two sides of message delivery:

1. Message delivery form the publishing client to the broker.

2. Message delivery from the broker to the subscribing client.

If the subscribing client defines a lower QoS than the publishing client, the broker transmits the message with the lower quality of service

QoS 0 - at most once

The minimal QoS level is zero. This service level guarantees a best-effort delivery. There is no guarantee of delivery. The recipient does not acknowledge receipt of the message and the message is not stored and re-transmitted by the sender. QoS level 0 is often called “fire and forget” and provides the same guarantee as the underlying TCP protocol.

QoS 1 - at least once

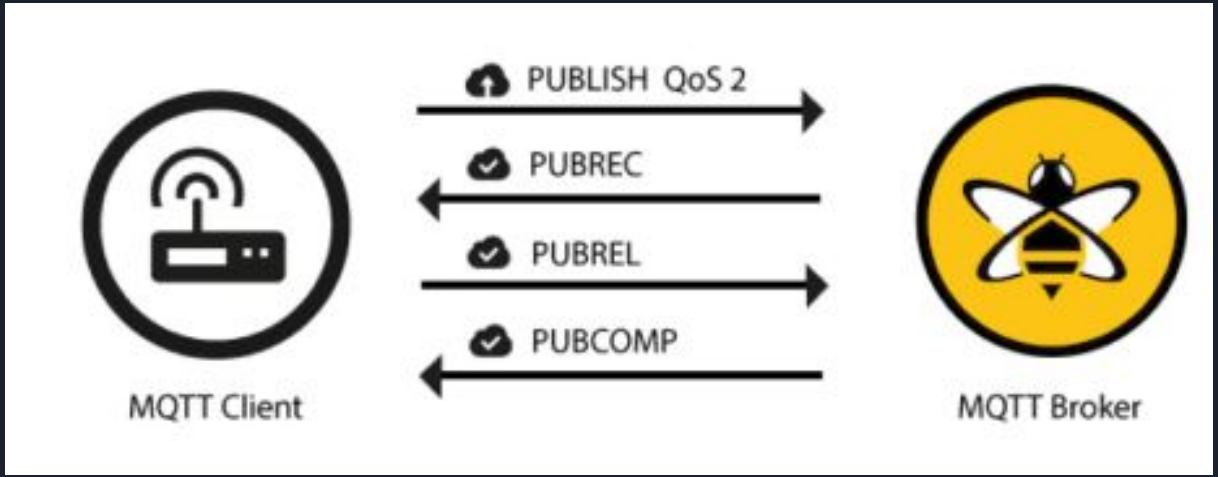
QoS level 1 guarantees that a message is delivered at least one time to the receiver. The sender stores the message until it gets a PUBACK packet from the receiver that acknowledges receipt of the message. It is possible for a message to be sent or delivered multiple times.

The sender uses the packet identifier in each packet to match the PUBLISH packet to the corresponding PUBACK packet. If the sender does not receive a PUBACK packet in a reasonable amount of time, the sender resends the PUBLISH packet. When a receiver gets a message with QoS 1, it can process it immediately. For example, if the receiver is a broker, the broker sends the message to all subscribing clients and then replies with a PUBACK packet.

If the publishing client sends the message again it sets a duplicate (DUP) flag. In QoS 1, this DUP flag is only used for internal purposes and is not processed by broker or client. The receiver of the message sends a PUBACK, regardless of the DUP flag.

QoS 2 - exactly once

QoS 2 is the highest level of service in MQTT. This level guarantees that each message is received only once by the intended recipients. QoS 2 is the safest and slowest quality of service level. The guarantee is provided by at least two request/response flows (a four-part handshake) between the sender and the receiver. The sender and receiver use the packet identifier of the original PUBLISH message to coordinate delivery of the message.



Persistent Session and Queuing Messages

To receive messages from an MQTT broker, a client connects to the broker and creates subscriptions to the topics in which it is interested. If the connection between the client and broker is interrupted during a non-persistent session, these topics are lost and the client needs to subscribe again on reconnect. Re-subscribing every time the connection is interrupted is a burden for constrained clients with limited resources. To avoid this problem, the client can request a persistent session when it connects to the broker. Persistent sessions save all information that is relevant for the client on the broker. The clientId that the client provides when it establishes connection to the broker identifies the session.

Persistent session

When the client connects to the broker, it can request a persistent session. The client uses a cleanSession flag to tell the broker what kind of session it needs:

- When the clean session flag is set to true, the client does not want a persistent session. If the client disconnects for any reason, all information and messages that are queued from a previous persistent session are lost.

- When the clean session flag is set to false, the broker creates a persistent session for the client. All information and messages are preserved until the next time that the client requests a clean session. If the clean session flag is set to false and the broker already has a session available for the client, it uses the existing session and delivers previously queued messages to the client.

Last Will and Testament

In MQTT, you use the Last Will and Testament (LWT) feature to notify other clients about an ungracefully disconnected client. Each client can specify its last will message when it connects to a broker. The last will message is a normal MQTTmessage with a topic, retained message flag, QoS, and payload. The broker stores the message until it detects that the client has disconnected ungracefully. In response to the ungraceful disconnect, the broker sends the last-will message to all subscribed clients of the last-will message topic.If the client disconnects gracefully with a correct

DISCONNECT message, the broker discards the stored LWT message.

Keep Alive

Keep alive ensures that the connection between the broker and client is still open and that the broker and the client are aware of being connected. When the client establishes a connection to the broker, the client communicates a time interval in seconds to the broker. This interval defines the maximum length of time that the broker and client may not communicate with each other.

Keep Alive

PINGREQ

The PINGREQ is sent by the client and indicates to the broker that the client is still alive. If the client does not send any other type of packets (for example, a PUBLISH or SUBSCRIBE packet), the client must send a PINGREQ packet to the broker. The client can send a PINGREQ packet any time it wants to confirm that the network connection is still alive. The PINGREQ packet does not contain a payload.

PINGRESP:

When the broker receives a PINGREQ packet, the broker must reply with a PINGRESP packet to show the client that it is still available. The PINGRESP packet also does not contain a payload.