Unique QoS Mechanisms of MQTT protocol

Telecommunication Networks and Systems final project

Kamil Chaj

MQTT, IoT, QoS

Abstract

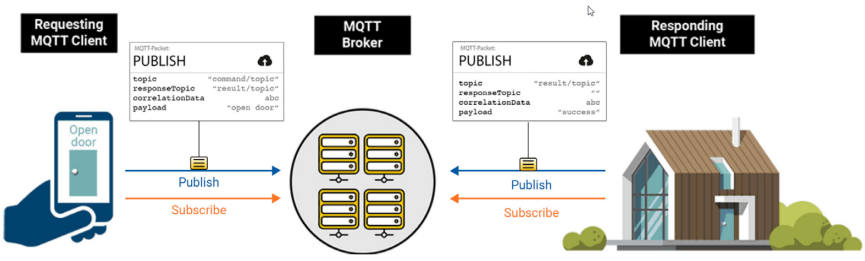
MQTT is lightweight publish/subscribe (PubSub) messaging protocol which allows for connection with many clients while ensuring high level of quality of service with the help of few unique mechanism.

1 Introduction to MQTT

MQTT is 5-7 layer protocol based on TCP and uses a binary message format for communication unlike HTTP or SMTP.

1.1 PubSub architecture

MQTT follows the publish-subscribe architecture which allows for decoupling of publisher from subscriber, clients are not aware of existence of each other, all messages are send to topic instead of directly to subscriber and broker handles everything in between.



Example: Smart door opening with a mobile device using MQTT

Both publish and subscribe packets constain packetId, topic name and qos level, but for publish packet there is obviously payload and two additional flags that will be mentioned in later sections.

PUBLISH	
packetId	2137
topicName	”counter/tick”
qos	1
retainFlag	false
payload	”Tick”
dupFlag	false

SUBSCRIBE	
packetId	2137
qos1	1
topic1	”counter/tick”
qos2	0
topic2	”counter/#/one”
qosN	2
topicN	”will/#”

Topic system is similar to file system, it is like a tree one topic can have multiple sub-topics. Wildcards can be used to subscribe to multiple topics.

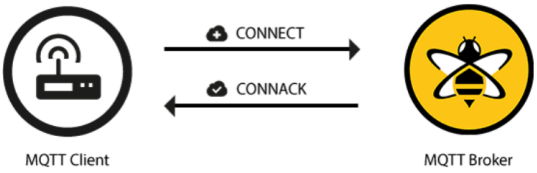
Example counter/value/one

Single-level wildcard counter/+/one

Multi-level wildcard counter/#

1.2 Connection to broker

To establish connection between broker and client, client must send CONNECT and receive CONNACK packet.



CONNECT packet contains, apart from basic information about client, configuration for persistent session, last will and lastly keep alive time which specifies maximum time between messages for broker to maintain connection with a client. To maintain connection with a broker, client sends PINGREQ packets to broker and receives PINGRESP

CONNECT	
clientId	"One"
cleanSession	false
username	"One"
password	"One"
lastWillTopic	"/will/one"
lastWillQos	2
lastWillMessage	"Client one offline"
lastWillRetain	false
keepAlive	60

2 Quality of service

The classical definition of quality of service is not accurate for the MQTT protocol, QoS is defined classically as parameters like latency, jitter, throughput, packet loss, or reliability, but in MQTT some of those parameters are not the main concern in usual situations that MQTT is used. QoS in MQTT is focused on the reliability of message delivery and handling connection loss.

2.1 Quality of Service levels

In MQTT publish and subscribe packet there is an option to set different levels of quality of service where additional resources can be traded to ensure delivery of a message. It is important to mention that QoS level works both when the broker and the client sends the message.

2.1.1 QoS 0 - At most once

The lowest QoS level requires only actual publish packet to be sent therefore is the cheapest but provides zero countermeasures from packet loss. QoS 1 is also commonly called "fire and forget".

Quality of Service level 0: delivery at most once

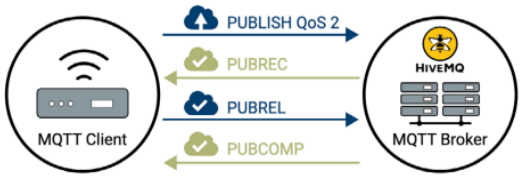
2.1.2 QoS 1 - At least once

Quality of service level 1 ensures that message will be delivered to a subscriber by sending the same publish packet repeatedly until the broker receives PUBACK packet. All packets that were not received at first try duplicate(DUP) flag is set.

Quality of Service level 1: delivery at least once

2.1.3 QoS 2 - Exactly once

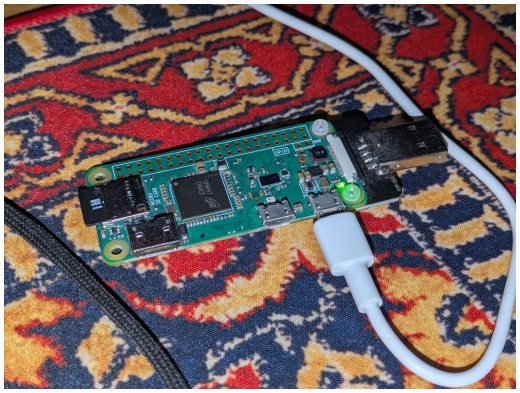
Messages with QoS level 2 come with three additional packets, PUBACK packet which works exactly like in QoS 1, PUBREL packet which is sent to receiver after sender receives PUBREL packet and PUBCOMP packet which ends message transmission.



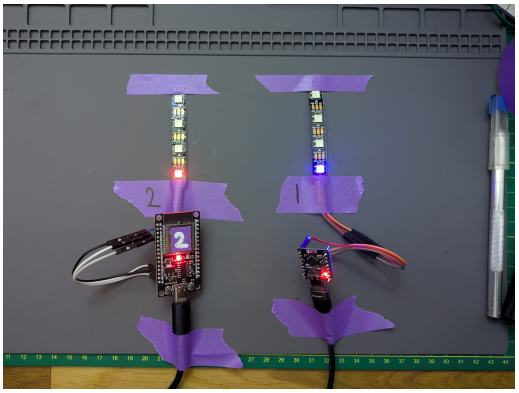
MQTT Quality of Service level 2: delivery exactly once

3 Demo setup

For the purpose of demonstration there are two client and broker connected to the same network over Wi-Fi. Clients are binary counters that display current state on LED strips, counter can be advanced by publishing to appropriate topic.



(a) Broker



(b) Clients

Topic structure:

- counter
  - value
    - one
    - two
  - color
    - one
    - two
  - tick
- will
  - one
  - two

Hardware:

- Raspberry Pi Zero W
- 2 ESP32 with WS28 LED strips

Software:

- Mosquitto MQTT broker
- PubSubClient - Arduino MQTT client

4 Persistent session

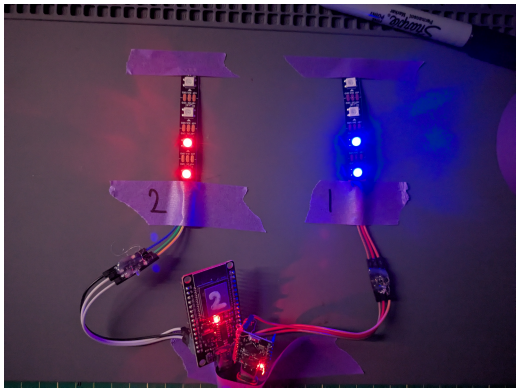
Persistent session is a feature of MQTT protocol that allows broker to remember sessions with clients and keep their topic subscriptions so they do not have to resubscribe every time they reconnect to the broker and ability to queue messages that were not received by a client and deliver them as soon as client is available.

To utilize this feature, client must connect to the broker with cleanSession flag false in CONNECT packet and in order for messages to be queued transmission must by at least QoS level 1.

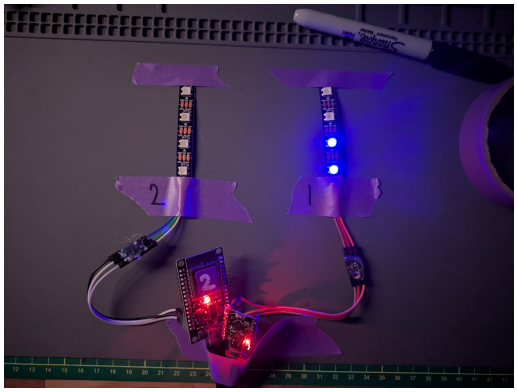
4.1 Demo

First step of persistent session demo is to publish a couple tick messages and record state of both counters, then disconnect both clients and publish the same amount of tick messages and reconnect both clients and record results.

```
mosquitto_pub -q 1 -t counter/tick -m Tick
```



(a) Messages sent while clients were online



(b) Messages set while clients were offline

Figure 2: Persistent session demo pictures

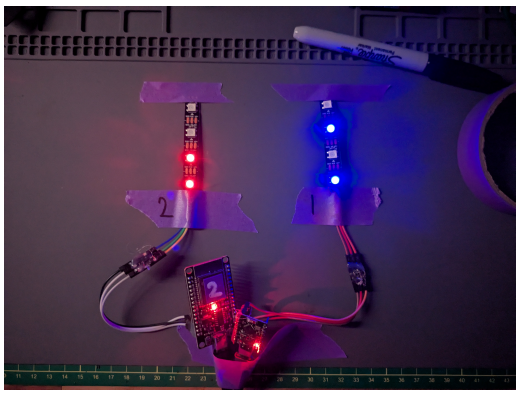
5 Retained messages

The retained message mechanism stores the latest message of a topic and provides it to all new subscribers to the topic. To enable this feature, retainFlag must be set in PUBLISH packet.

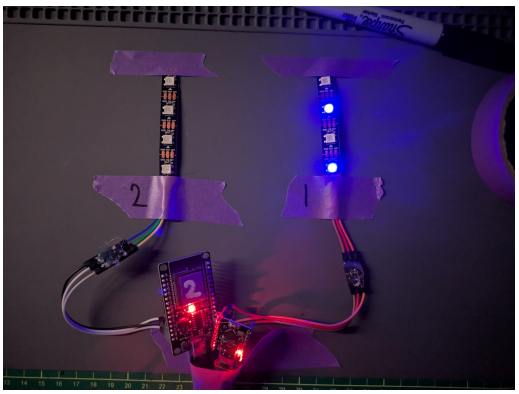
5.1 Demo

For this demo publish different values for both counter but for one of them set retain flag, then disconnect and reconnect both clients and record state of both counters.

```
mosquitto_pub -q 1 -t counter/value/one -m 3 -r
mosquitto_pub -q 1 -t counter/value/two -m 5
```



(a) Clients before disconnection



(b) Clients after disconnection

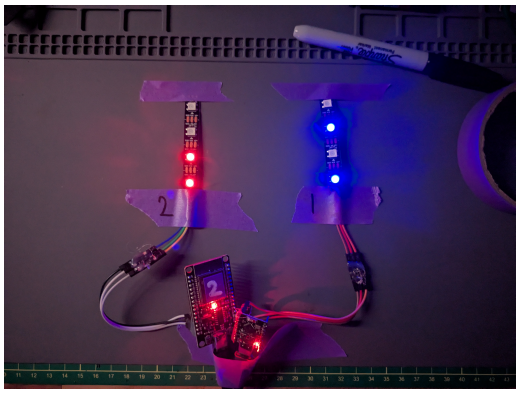
Figure 3: Retained messages demo pictures

6 Last will and Testament (LWT)

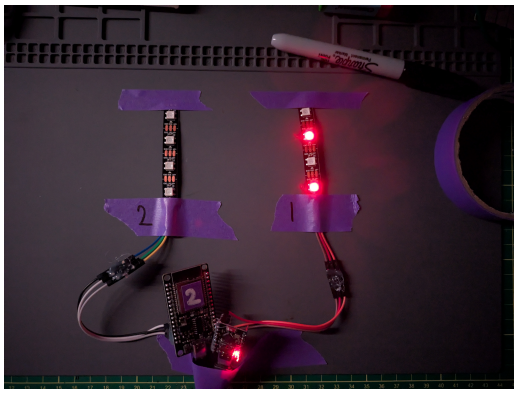
This feature allows broker to inform other clients about lost connection with a client by publishing a message to a topic which are specified in CONNECT packet, last will message can be sent with any QoS level and can also be retained.

6.1 Demo

To demonstrate LWT, simply disconnect one of the clients and wait for the broker to recognize that one of the clients is offline and record color of the connected counter.



(a) Both client online



(b) Client two disconnected

```
Client (null) received PUBLISH (d0, q0, r0, m0, 'counter/color/one', ... (3 bytes))
255
Client (null) received PUBLISH (d0, q1, r0, m10, 'will/two', ... (18 bytes))
Client (null) sending PUBACK (m10, rc0)
Client two offline
Client (null) received PUBLISH (d0, q0, r0, m0, 'counter/color/one', ... (8 bytes))
16711680
```

(c) Message log

Figure 4: Last will demo pictures

7 Conclusions

MQTT provides many features which greatly improve quality of service while using minimal resources by offloading many tasks to the broker. Personally, I found using MQTT easy and mostly problem-free, all features mentioned happen behind the scenes, and only thing of concern for the engineer is setting few bits. This was my first encounter with PubSub architecture, at first it was

confusing but once I started using it, it was not that difficult. In the future, MQTT will definitively be one of the options for me when it comes to communication protocols for embedded systems.

A Resources

Demo source code https://github.com/kamilix2003/MQTT-demo

HiveMQ MQTT Essentials eBook https://www.hivemq.com/mqtt/

PubSubClient Arduino client for MQTT https://pubsubclient.knolleary.net/

Mosquitto MQTT broker https://mosquitto.org/

Random Nerd Tutorials used for initial demo setup Broker, Client