Systems of Systems (SYOSY)

M.Sc. In Critical Computing Systems Engineering

ISEP/IPP – 2021/22, 2nd semester

Assignment 1:

M2M Messaging Protocols

Pedro Santos

Outline

Introduction to M2M Messaging Protocols

M2M Messaging Protocols

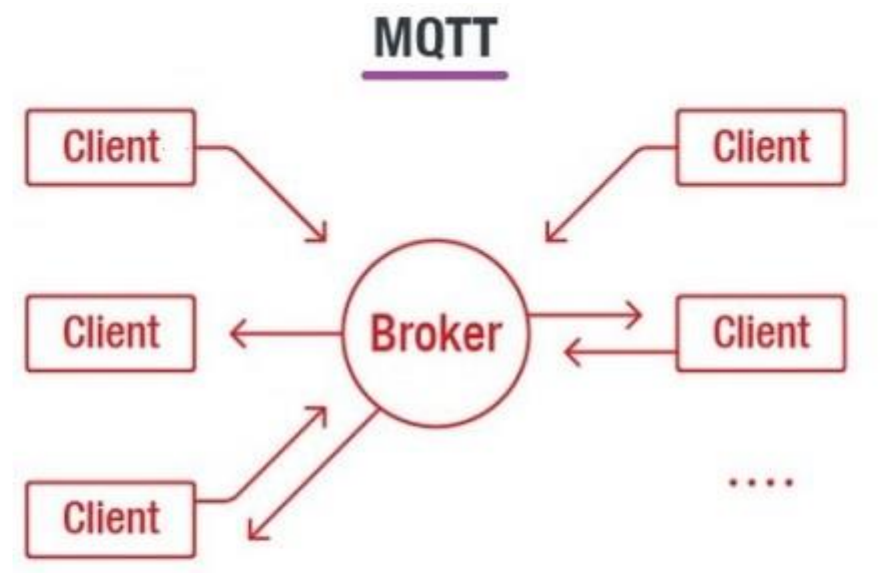
- Machine-to-Machine (M2M) communications are becoming more and more relevant, to enable inter-machine communication
- Dedicated messaging protocols have been proposed as middleware to this type of communication
- Examples
  - Message Queue Telemetry Transport (MQTT)
  - Constrained Application Protocol (CoAP)

Review of MQTT

Message Queue Telemetry Transport (MQTT)

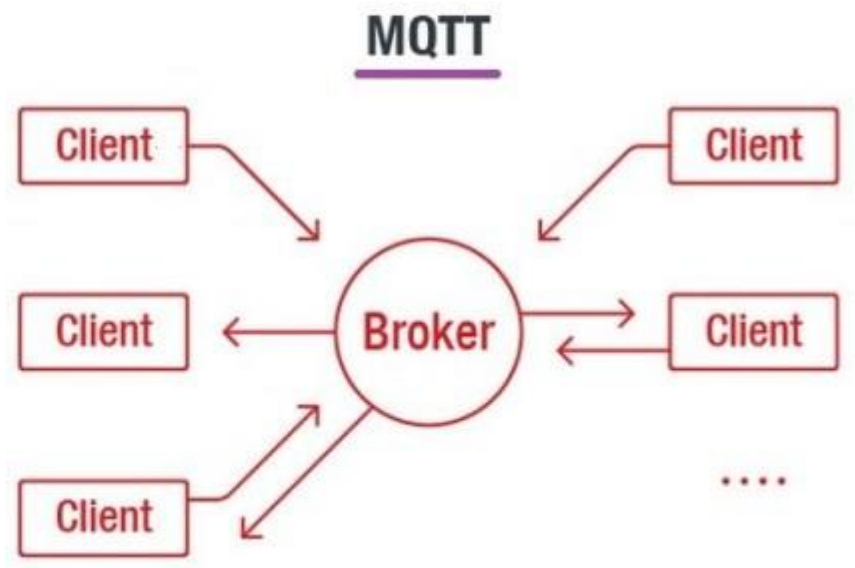
- MQTT is an OASIS standard messaging (and an ISO recommendation ISO/IEC 20922). protocol for the Internet of Things (IoT) and was designed by IBM.
- Lightweight machine-to-machine communication protocol for topic-based publish-subscribe architectures.
- Oriented for connecting remote devices with small code footprint and minimal network bandwidth.

- Publisher-subscriber paradigm:



Publisher-Subscriber Paradigm

- Clients do not have addresses like in email systems, and messages are not sent to clients.
- **Messages are published to a broker on a topic**. For example, a publisher might send a message temp: 22.5 on a topic heating/thermostat/living-room.
- **The job of an MQTT broker is to filter messages based on topic, and then distribute them to subscribers**.
- A client can receive these messages by subscribing to that topic on the same broker
- There is no direct connection between a publisher and subscriber.
- All clients can publish (broadcast) and subscribe (receive).
- MQTT brokers do not normally store messages.



Connections and Client IDs

Connections

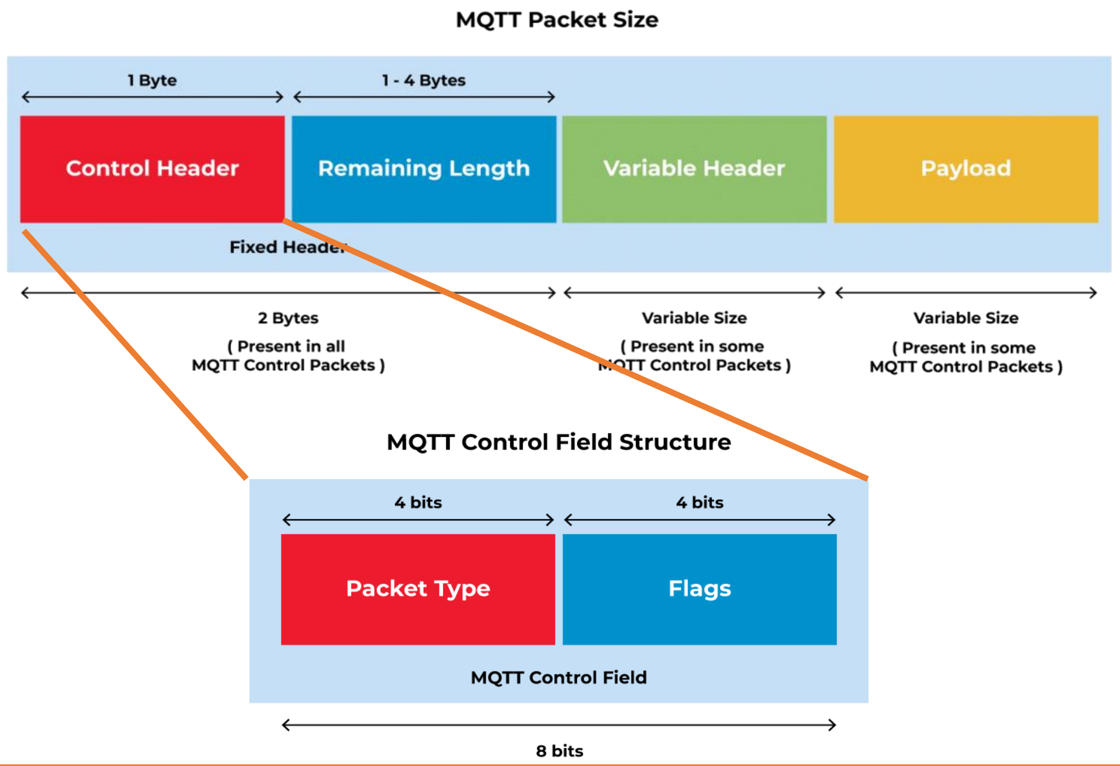
- Connections are acknowledged by the broker using a Connection acknowledgement message.
- ***You cannot publish or subscribe unless you are connected.***

Client Name or Client ID

- All clients are required to have a client name or ID.
- The client name is used by the MQTT broker to track subscriptions etc.
- Client names must also be unique.
- If you attempt to connect to an MQTT broker with the same name as an existing client then the existing client connection is dropped.

MQTT Packet Format

- It requires a fixed header of 2 bytes.
- The first byte is the control header (where the first 4 bits are message type and the other 4 bits are control flags) and the packet length goes in the second byte, extending for 3 more bytes is necessary.



Message Type:

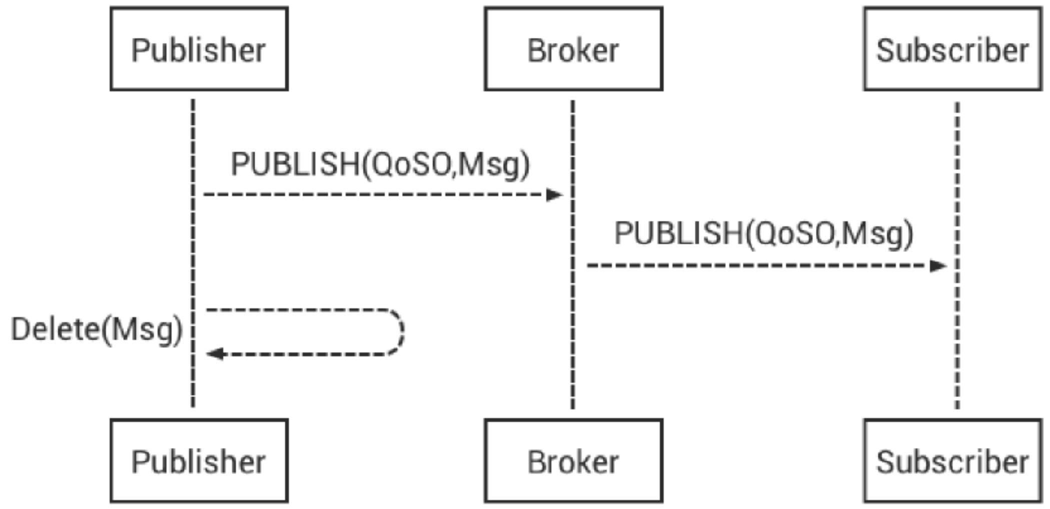
Message type	Value	Description	Fixed Header
Reserved	0	Reserved	Present
CONNECT	1	Client connect request to server or broker	Present
CONNACK	2	Connect request acknowledgment	Present
PUBLISH	3	Publish message	Present
PUBACK	4	Publish acknowledgment	Present
PUBREC	5	Publish receive	Present
PUBREL	6	Publish release	Present
PUBCOMP	7	Publish complete	Present
SUBSCRIBE	8	Client subscribe request	Present
SUBACK	9	Subscribe request acknowledgment	Present
UNSUBSCRIBE	10	Unsubscribe request	Present
UNSUBACK	11	Unsubscribe acknowledgment	Present
PINGREQ	12	PING request	Present
PINGRESP	13	PING response	Present
DISCONNECT	14	Client is disconnecting	Present
Reserved	15	Reserved	Present

Header Flags Structure in details:

3	2	1	0	bit
DUP	**QOS**	**QOS**	**RETAIN**	Header Flags

Reliability in MQTT: QoS modes

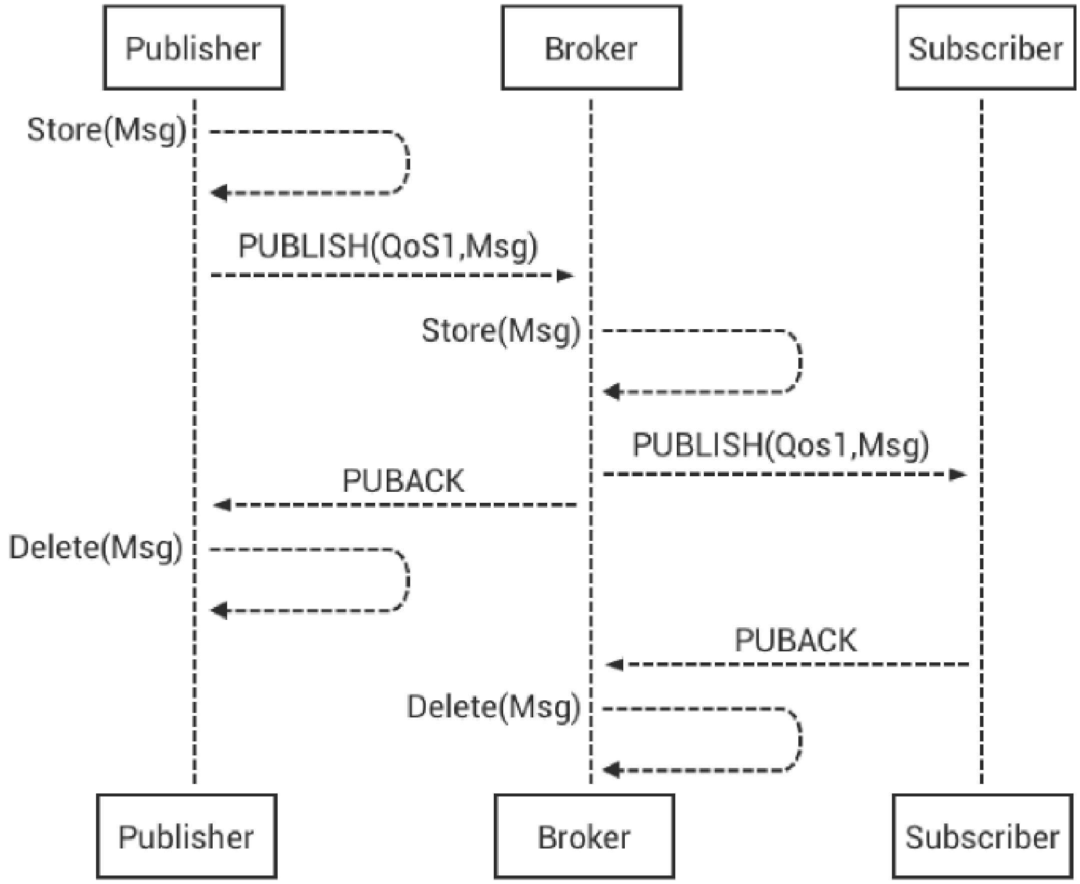
- MQTT offers three levels of Quality of Service (QoS) for reliable message delivery: QoS0, QoS1, and QoS2.
- QoS0 (commonly called Fire and Forget or At most once) offers only a best-effort delivery. There is no guarantee of delivery and the recovery effort is null or minimal.
- The recipient does not 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.
- Use QoS 0 when …
  - **You have a completely or mostly stable connection between sender and receiver.** A classic use case for QoS 0 is connecting a test client or a front end application to an MQTT broker over a wired connection.
  - **You don't mind if a few messages are lost occasionally.** The loss of some messages can be acceptable if the data is not that important or when data is sent at short intervals
  - **You don't need message queuing.** Messages are only queued for disconnected clients if they have QoS 1 or 2 and a persistent session.



QoS0 – *Fire and Forget*

MQTT QoS1 mode

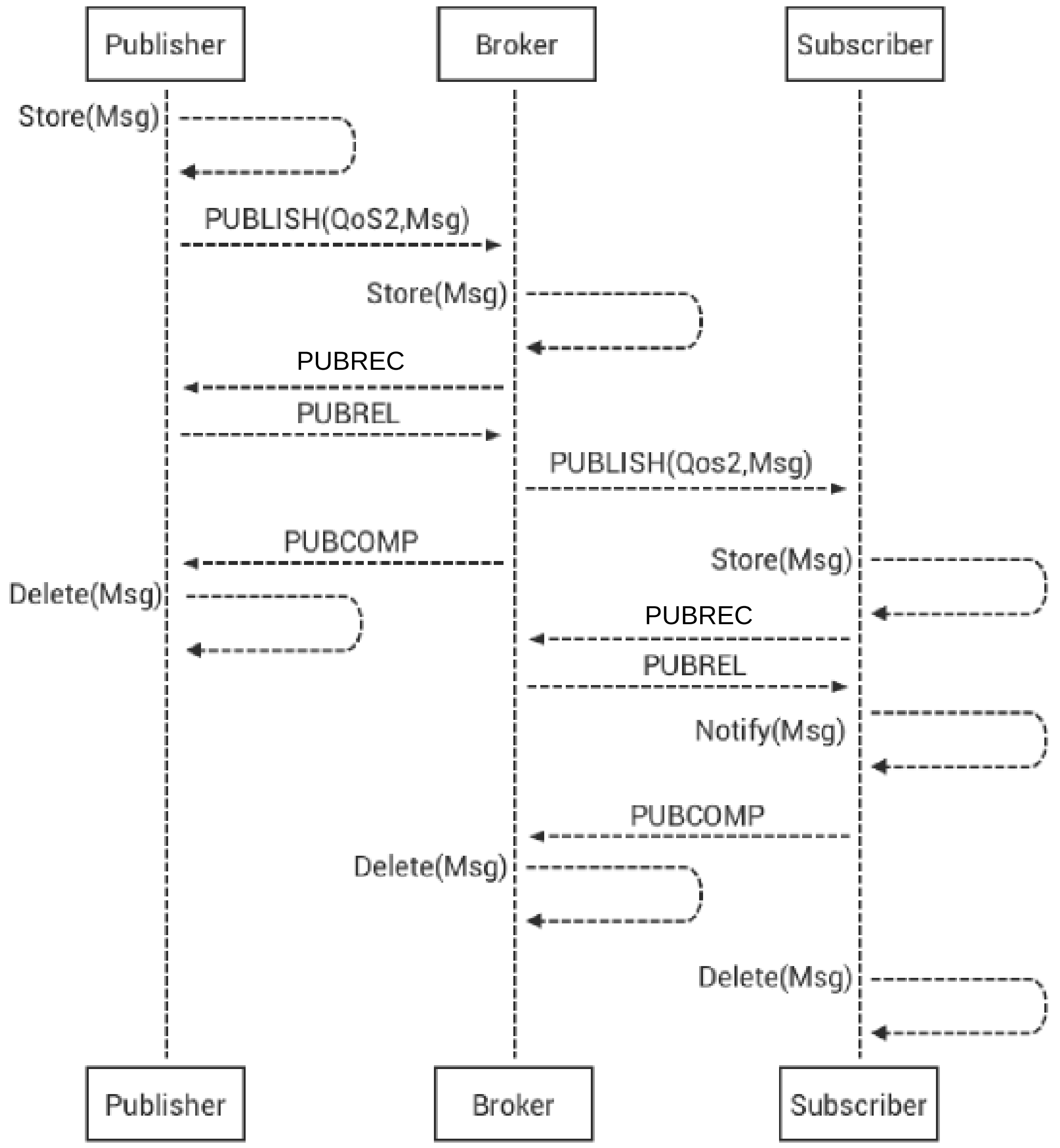
- **QoS1 (At least Once) guarantees that a message is delivered at least one time to the receiver**. The sender stores the message until it gets a PUBACK packet from the receiver that acknowledges the message.
    1. The sender uses the packet identifier in each packet to match the PUBLISH packet to 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.
    2. 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.
    3. 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.
- Use QoS 1 when …
    - **You need to get every message and your use case can handle duplicates.** QoS level 1 is the most frequently used service level because it guarantees the message arrives at least once but allows for multiple deliveries. Of course, your application must tolerate duplicates and be able to process them accordingly.
    - You can't bear the overhead of QoS 2. **QoS 1 delivers messages much faster than QoS 2.**



QoS1 - At least once

MQTT QoS2 mode

- QoS2 offers the highest reliability - Exactly Once. This level guarantees that each message is received only once by the intended recipients.
- This guarantee is provided by at least two request/response flows (a four-part handshake) between the sender and the receiver.
  1. Receiver gets a QoS 2 PUBLISH packet from a sender, and replies with a PUBREC packet that acknowledges the PUBLISH packet. If the sender does not get a PUBREC packet from the receiver, it sends the PUBLISH packet again with a duplicate (DUP) flag.
  2. Once the sender receives a PUBREC packet from the receiver, the sender can discards the initial PUBLISH packet. The sender stores the PUBACK packet from the receiver and responds with a PUBREL packet.
  3. After the receiver gets the PUBREL packet, it can discard all stored states and answer with a PUBCOMP packet. After the sender receives the PUBCOMP packet, the packet identifier of the published message becomes available for reuse.
- Use QoS 2 when …
  - **It is critical to your application to receive all messages exactly once**. This is often the case if a duplicate delivery can harm application users or subscribing clients. Be aware of the overhead and that the QoS 2 interaction takes more time to complete.



QoS2 – Exactly Once

Persistency & Retained Messages

Persistence

- Using a persistent connection, **the broker will store subscription information, and undelivered messages for the client**.
- In order for the broker to store session information for a client, a client id must be used.
- When a client connects to broker it indicates whether the connection is **persistent** or not, by setting the 'clean-session' flag to FALSE or TRUE respectively, in the CONNECT packets.
    - **In command-line, clean session is the default**. In the *mosquitto_sub* command, **add flag '`–c`' ('`--disable-clean-session`') for a persistent connection.**
- However it is important to realise that not all messages will be stored for delivery, as the quality of service, of the subscriber and publisher has an effect.

Retained Messages

- If a publisher publishes a message to a topic and no one is subscribed to that topic, the message is simply discarded by the broker.
- However the publisher can tell the broker to keep the last message on that topic by setting the "retained" message flag.
- This can be very useful, as for example, if you have sensor publishing its status only when changed e.g. Door sensor. What happens if a new subscriber subscribes to this status? Without retained messages the subscriber would have to wait for the status to change before it received a message.
- What is important to understand is that only one message is retained per topic. The next message published on that topic replaces the last retained message for that topic.

QoS modes in MQTT

Steps

1. Install MQTT Broker
2. Configure MQTT Broker
3. Connect to RPi for inspecting traffic
4. Test QoS modes

1./2. Install & Configure MQTT Broker

1. Follow instructions here:
   - https://appcodelabs.com/introduction-to-iot-build-an-mqtt-server-using-raspberry-pi
   - (note: to test publish & subscribe you must have two SSH connections)

2. Configure MQTT Broker:
   - In */etc/mosquitto/mosquitto.conf*, add
     - `allow_anonymous true`
     - `listener 1883`

3. Inspect different types of MQTT QoS

1. Connect to RPI via VNC (if in Windows)
2. Start Wireshark: `sudo wireshark`
3. Apply filter ‘MQTT’
4. Start

```
sudo systemctl stop mosquitto
sudo pkill mosquitto
sudo mosquitto -v –c /etc/mosquitto/mosquitto.conf
sudo tail –f /var/log/mosquitto/mosquitto.log
```

4. Test QoS modes

Common steps:

1. Initialise the client, requesting or not a ‘clean_session’
2. Subscribe to a topic with QoS set
3. Disconnect.
4. Publish to the topic that the client subscribed to with QOS set.
5. Reconnect client
6. Make note of any messages received

4. Test QoS modes

Test 1 – QoS=0 & no Persistence

- Publish / Subscribe QoS: 0
- clean_session=TRUE (‘-c’ flag is omitted, so default behaviour is non-persistent connection)
- Steps:
  1. Start Pub: `mosquitto_pub -h localhost -t "test/message" -m "MESSAGE 1" -q 0`
  2. Start Sub: `mosquitto_sub -h localhost -i Sub1 -t "test/message" -q 0`
- Result: ?

Test 2 – QoS=0 & Persistence

- Publish / Subscribe QoS: 0
- “clean_session”=FALSE (‘-c’ flag indicates persistent connection)
- Steps:
  1. Start Pub: `mosquitto_pub -h localhost -t "test/message" -m "MESSAGE 2" -q 0`
  2. Start Sub: `mosquitto_sub -h localhost -i Sub1 -t "test/message" -q 0 –c`
- Result: ?

4. Test QoS modes

Test 3 – QoS=1/2 & Persistence

- Publish / Subscribe QoS: 1 or 2
- “clean_session”=FALSE
- Steps:
  1. Start Pub: `mosquitto_pub -h localhost -t "test/message" -m "MESSAGE 3" -q 1/2`
  2. Start Sub: `mosquitto_sub -h localhost -i Sub1 -t "test/message" -q 1/2 -c`
- Result: ?

Test 4 – Different Pub/Sub QoS

- Publish QoS: 1; Subscribe QoS: 0
- “clean_session”=FALSE
- Steps:
  1. Start Pub: `mosquitto_pub -h localhost -t "test/message" -m "MESSAGE 4" -q 1`
  2. Start Sub: `mosquitto_sub -h localhost -i Sub1 -t "test/message" -q 0 -c`
- Result: ?

4. Test QoS modes

Test 5 – Different Pub/Sub QoS

- Publish QoS: 0; Subscribe QoS: 1
- “clean_session”=FALSE
- Steps:
  1. Start Pub: `mosquitto_pub -h localhost -t "test/message" -m "MESSAGE 4" -q 1`
  2. Start Sub: `mosquitto_sub -h localhost -i Sub1 -t "test/message" -q 0 -c`
- Result: ?

End of Part 1

References

- Steve Cope. “MQTT Clean Sessions and QOS Examples”. *http://www.steves-internet-guide.com/mqtt-clean-sessions-example/*