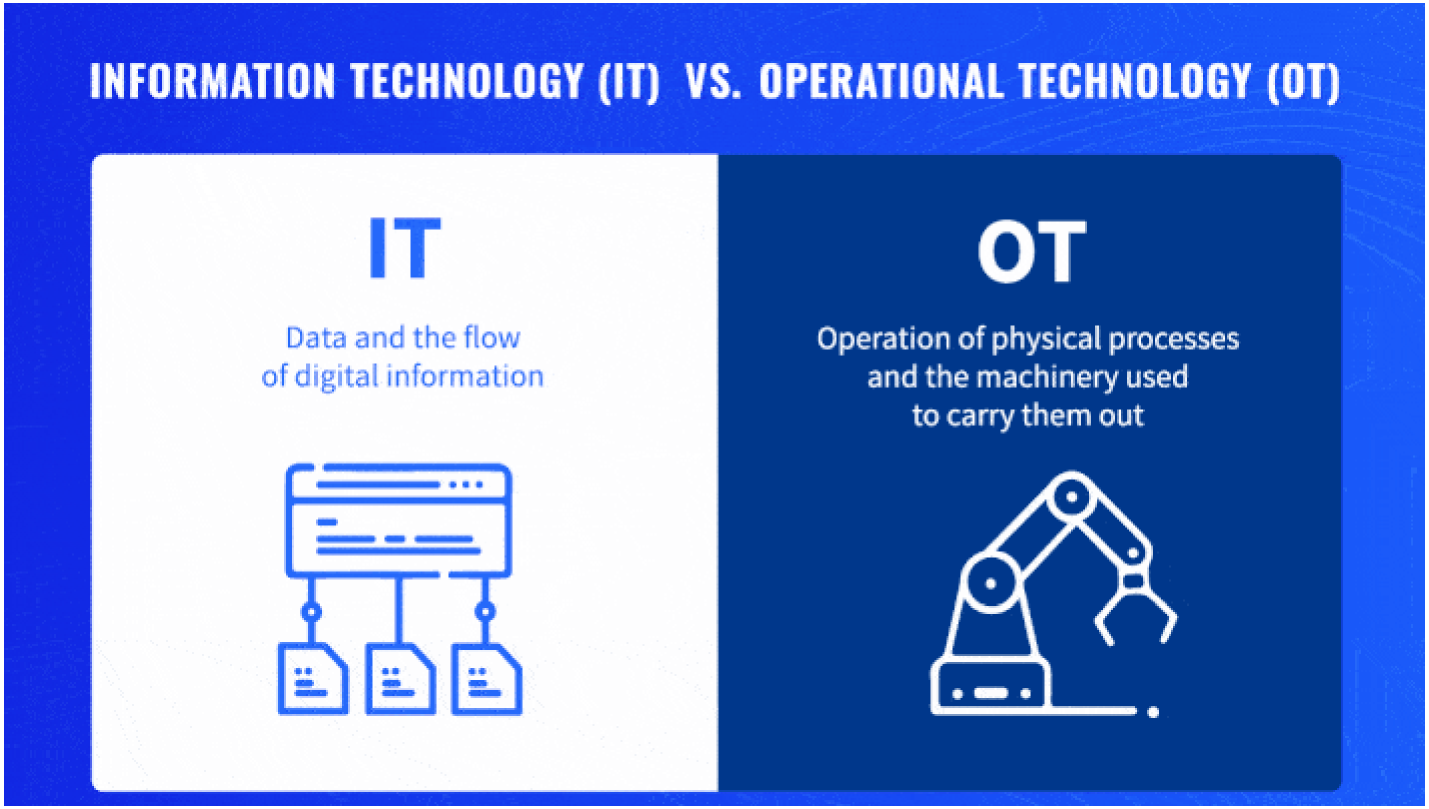
MQTT

Message Queuing Telemetry Transport (MQTT)



INFORMATION TECHNOLOGY (IT) VS. OPERATIONAL TECHNOLOGY (OT)

IT

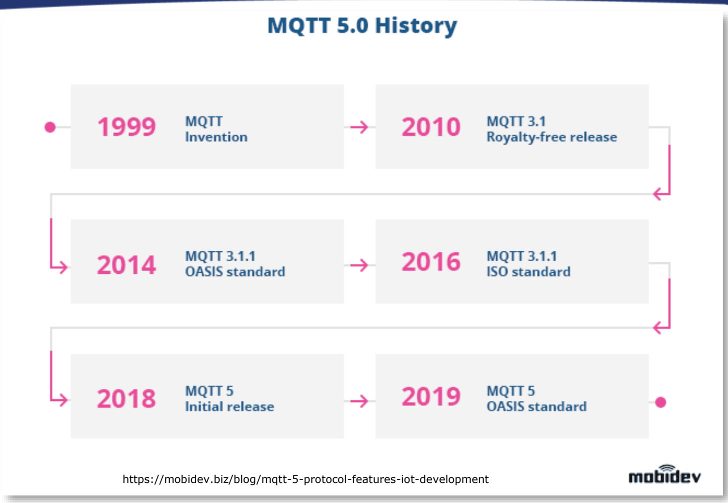
Data and the flow
of digital information

OT

Operation of physical processes
and the machinery used
to carry them out

MQTT

What is it?



MQTT 5.0 History

1999 MQTT Invention → 2010 MQTT 3.1 Royalty-free release

2014 MQTT 3.1.1 OASIS standard → 2016 MQTT 3.1.1 ISO standard

2018 MQTT 5 Initial release → 2019 MQTT 5 OASIS standard

https://mobidev.biz/blog/mqtt-5-protocol-features-iot-development

mobidev

- Invented 1999
- Is a dedicated protocol for IoT (Internet of Things)
- Is an OASIS Standard (https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf) with ~137 pages
- Latest version is 5.0 (new Features for Cloud Connectivity), 3.1.1 is currently heavily used
- Goal: establish a **lightweighted protocol for IoT-devices and embedded systems**
- Today, PLCs support MQTT (besides others): so it can be used for communicating from a higher-level language with PLCs

“MQTT is a Client Server publish/subscribe messaging transport protocol. It is **light weight, 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**.“

https://www.iso.org/standard/69466.html

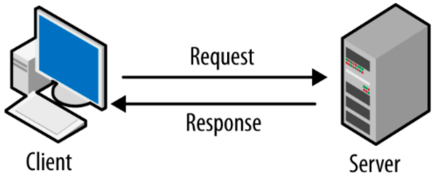
It has a smaller footprint than HTTP

Related Technologies

Publisher-Subscriber Principle

- HTTP,..: Client sends request to Server, the server answers

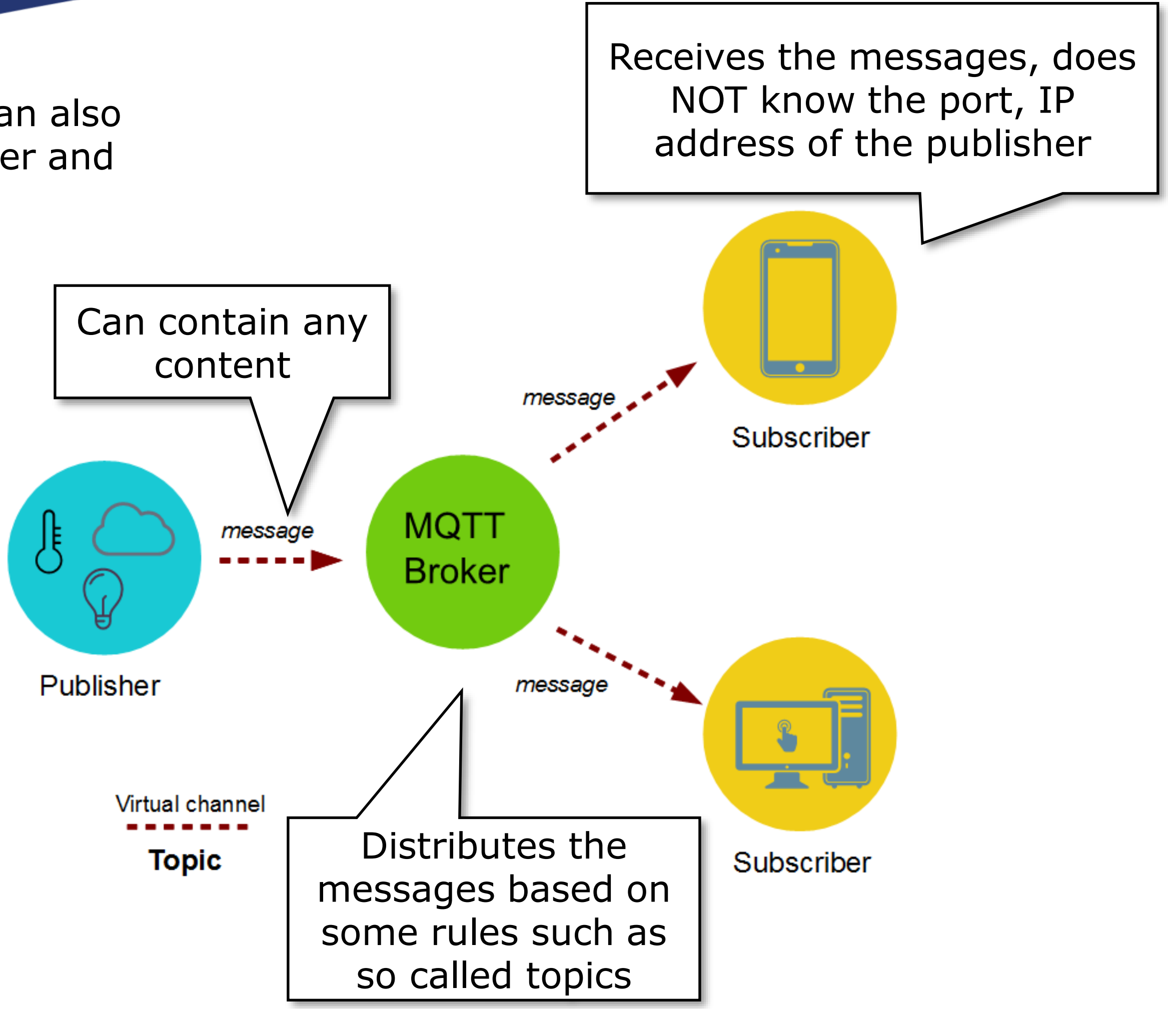
→ Client-Server Principle



https://madooei.github.io/cs421_sp20_homepage/assets/client-server-1.png

- Publisher –Subscriber Principles decouples the server from the client.
  - Publisher: sends messages
  - Subscriber: receives messages
  - Broker: the new party acting between publisher and subscriber

A publisher can also be a subscriber and vice versa!



Receives the messages, does NOT know the port, IP address of the publisher

Can contain any content

Distributes the messages based on some rules such as so called topics

https://www.cadlog.es/wp-content/uploads/2017/11/mqtt_publisher_subscriber.png

Message Queue vs MQTT

- **Message Storage:** A Message Queue stores messages until consumed (all messages are always consumed). **In MQTT there could be published messages which are not subscribed** (never consumed)
- **Multiple Subscribers:** In a message queue usually a message is only consumed once. In MQTT there a multiple subscribers, the **messages can be subscribed multiple times**
- Creation: Messages Queues have to be created explicitly while in MQTT **topics** can simply be **created by „publishing to them“**

Lets try with MQTT Explorer…

…and via cmd (see cmd)!

Also, look at the config of Mosquitto!

Implementations

Implications of the PubSub Principle

- Publisher does not know subscribers
- Asynchronous: When publisher sends a message, the subscriber needs not to be connected to the broker
- The broker gets all messages and can filter: it decides which subscriber gets which message

  Filtering is based:
  - Topic
  - Msg Content (Problems with encryption)

  This architecture is considered as more scalable as a traditional client-server principle:
  - Broker can cache messages
  - Clustering of Brokers is possible

Message vs Event

Just for discussion: we will not make a distinction between those two terms

Message:

- The sending component expects the destination component to process the message content in a certain way

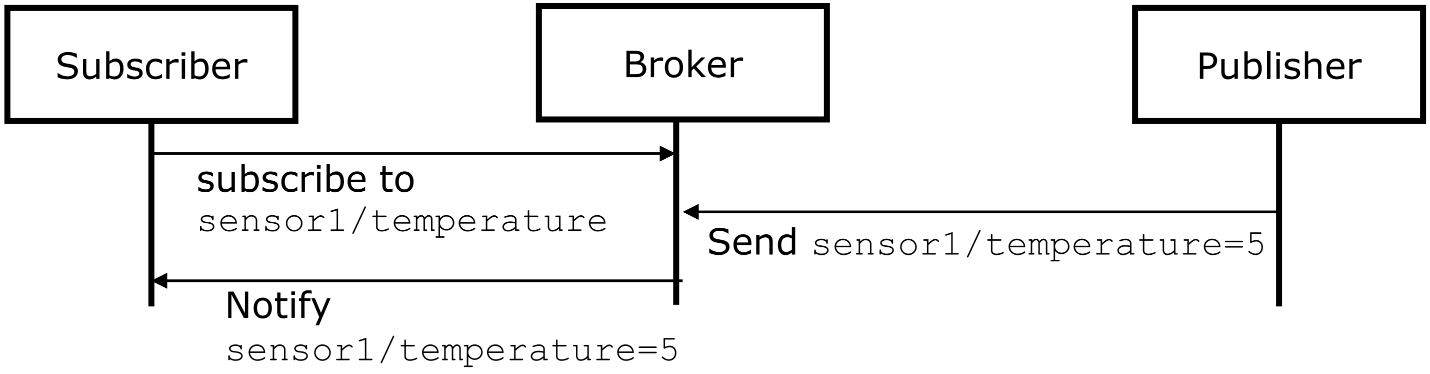
Events:

- The publisher of the event has no expectation about the action a receiving component takes.
- An event is a lightweight notification that indicates that something happened.

https://learn.microsoft.com/en-us/training/modules/choose-a-messaging-model-in-azure-to-connect-your-services/2-choose-whether-to-use-message-queues-or-events

Topics

- Subscribers do not subscribe to the complete system (like to a message queue), instead they subscribe to topics
- Topics are Endpoints where messages are sent to
  Example: `sensor1/temperature`
- Subscribers that subscribe to such an topic will receive all messages published to it



Subscriber

Broker

Publisher

subscribe to `sensor1/temperature`

Send `sensor1/temperature=5`

Notify `sensor1/temperature=5`

Topic Wildcard

- Clients can also subscribe to unkown topics with wildcards
- # stands for any topic or subtopic (multi level)
- + stands for exactly one topic (single level)

Example:

If you subsribe to `sensor1/#` you will receive published values of the topics `sensor1/temperature` and `sensor1/wind`

Topic Wildcard

You subscribe to `sensor1/+/#`

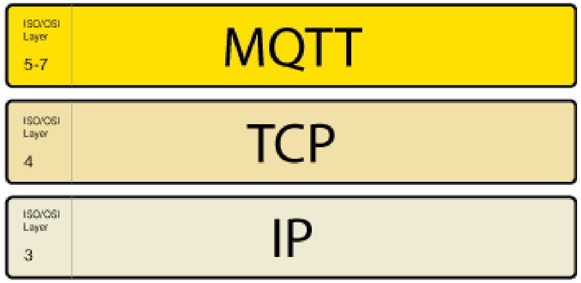
Which messages will you receive?

- `sensor1/temperature`
- `sensor1`
- `sensor1/temperature/front`
- `sensor1/temperature/front/median`

Wildcard.ipynb

Can you subscribe to sensor1/+/front?

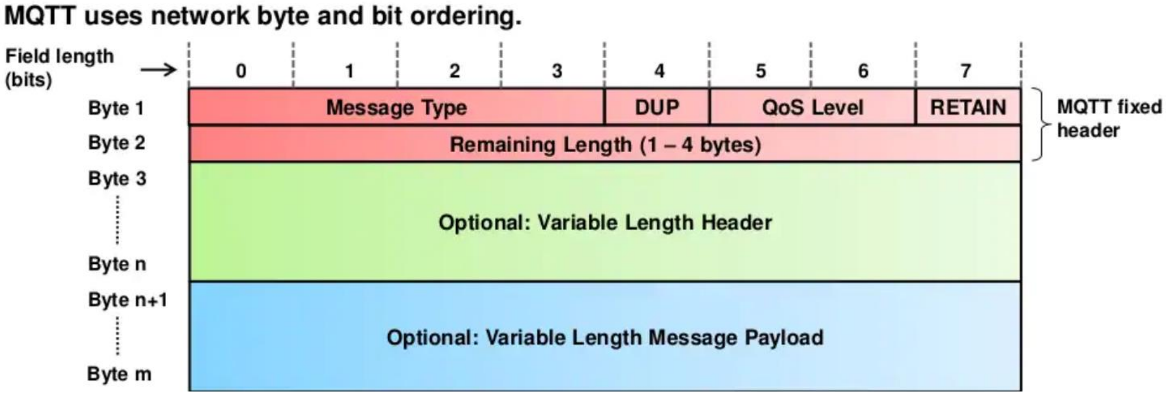
MQTT Protocol Stack



ISO/OSI Layer 5-7 MQTT

ISO/OSI Layer 4 TCP

ISO/OSI Layer 3 IP



MQTT uses network byte and bit ordering.

Field length (bits) → 0 1 2 3 4 5 6 7

Byte 1: Message Type (bits 0–3) | DUP (bit 4) | QoS Level (bits 5–6) | RETAIN (bit 7)

Byte 2: Remaining Length (1 – 4 bytes)

MQTT fixed header

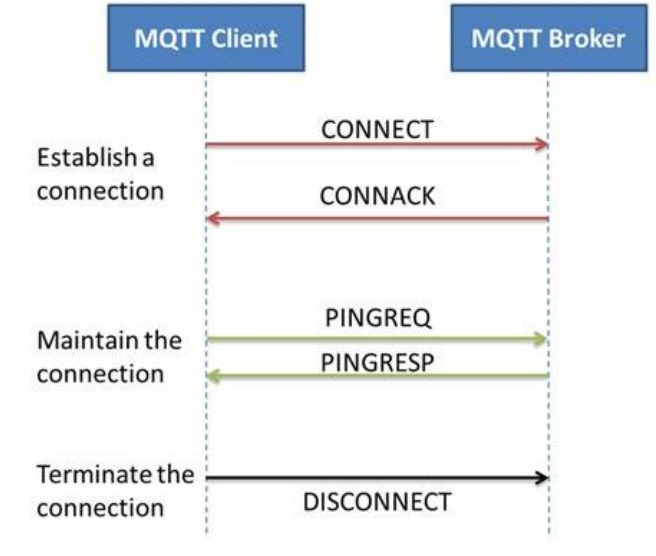
Byte 3 … Byte n: Optional: Variable Length Header

Byte n+1 … Byte m: Optional: Variable Length Message Payload

The minimum MQTT Package are 2 bytes (8 bytes in http)

https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/
https://www.geeksforgeeks.org/difference-between-mqtt-and-http-protocols/
https://pt.slideshare.net/PeterREgli/mq-telemetry-transport

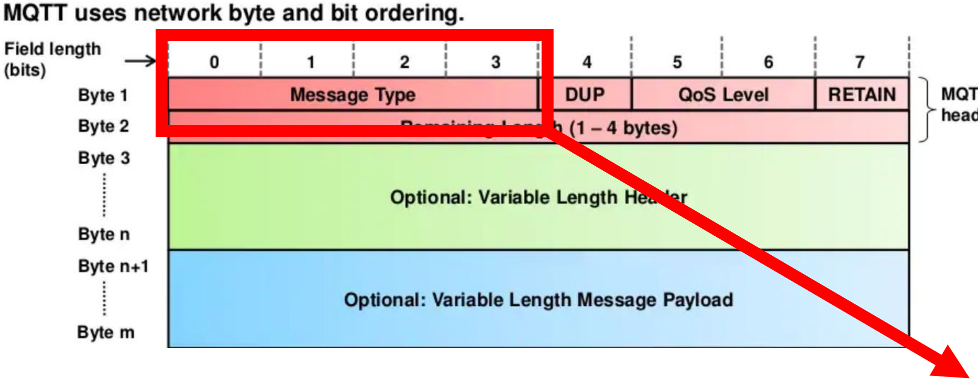
MQTT: Establish connection with broker



"After establishing a connection between the MQTT client and the broker, the first packet must be a CONNECT packet. The CONNECT packet only needs to be sent once over the network connection. **The second CONNECT packet sent by the MQTT client is ignored and disconnected"**

https://www.bevywise.com/blog/understanding-mqtt-protocol-packet-format/#:~:text=The%20Remaining%20Length%20is%20the,for%20values%20up%20to%20127.

Kodali et. al (2016): MQTT based home automation system using ESP8266


MQTT uses network byte and bit ordering.
Field length (bits)
0
1
2
3
4
5
6
7
Byte 1
Byte 2
Byte 3
Byte n
Byte n+1
Byte m
Message Type
DUP
QoS Level
RETAIN
Remaining Length (1 – 4 bytes)
Optional: Variable Length Header
Optional: Variable Length Message Payload
MQTT fixed header

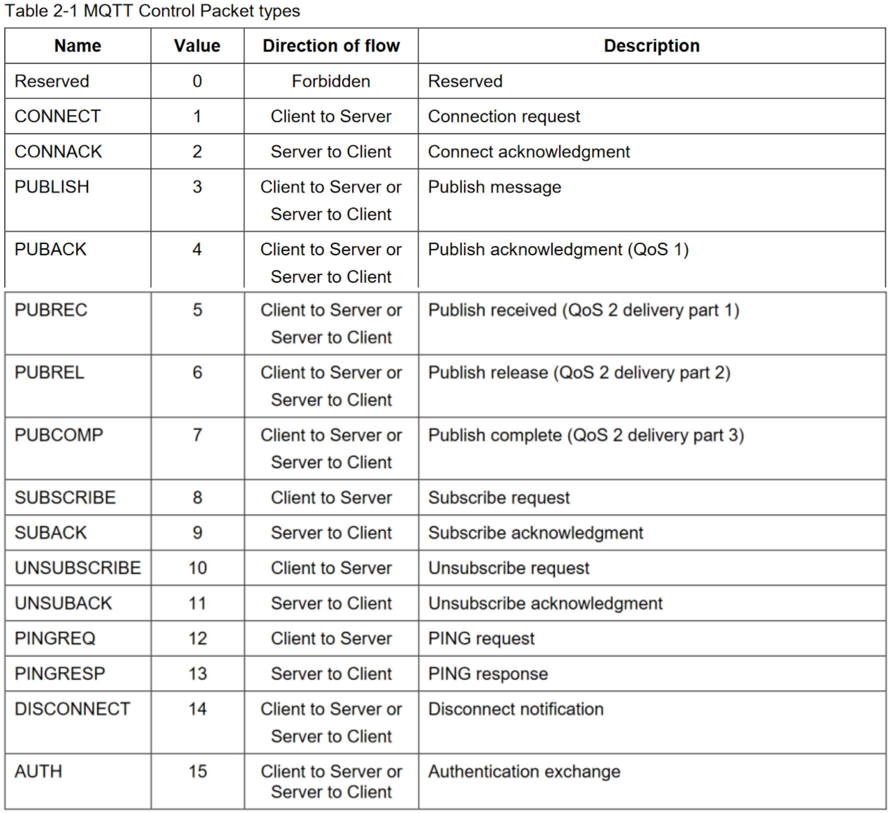


Table 2-1 MQTT Control Packet types

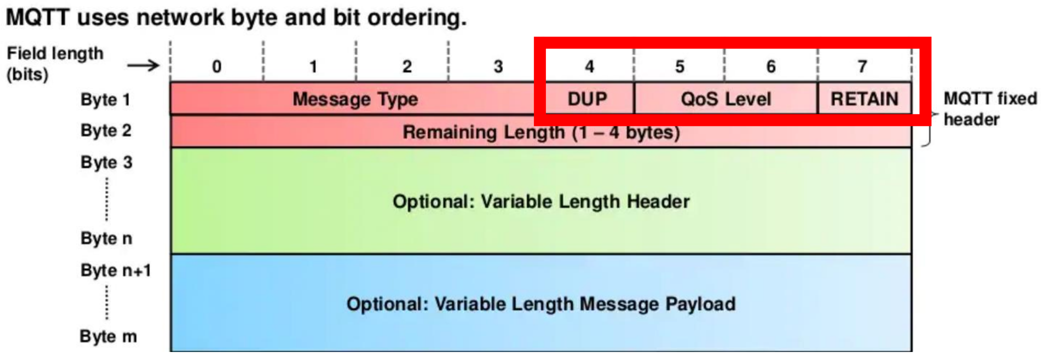
Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

First four bits are used to indicate the message type (2^4=16 possible message types that are currently supported)

Connect and ConnAck are two of that types

Also PingReq and PringResponse are two of that types

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf


MQTT uses network byte and bit ordering.
Field length (bits)
0
1
2
3
4
5
6
7
Byte 1
Byte 2
Byte 3
Byte n
Byte n+1
Byte m
Message Type
DUP
QoS Level
RETAIN
Remaining Length (1 – 4 bytes)
MQTT fixed header
Optional: Variable Length Header
Optional: Variable Length Message Payload

The remaining four bits are used for PUBLISH messages:

- DUP = Duplicate delivery of a PUBLISH packet
  The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH packet.

- QoS = PUBLISH Quality of Service

- RETAIN = PUBLISH retained message flag

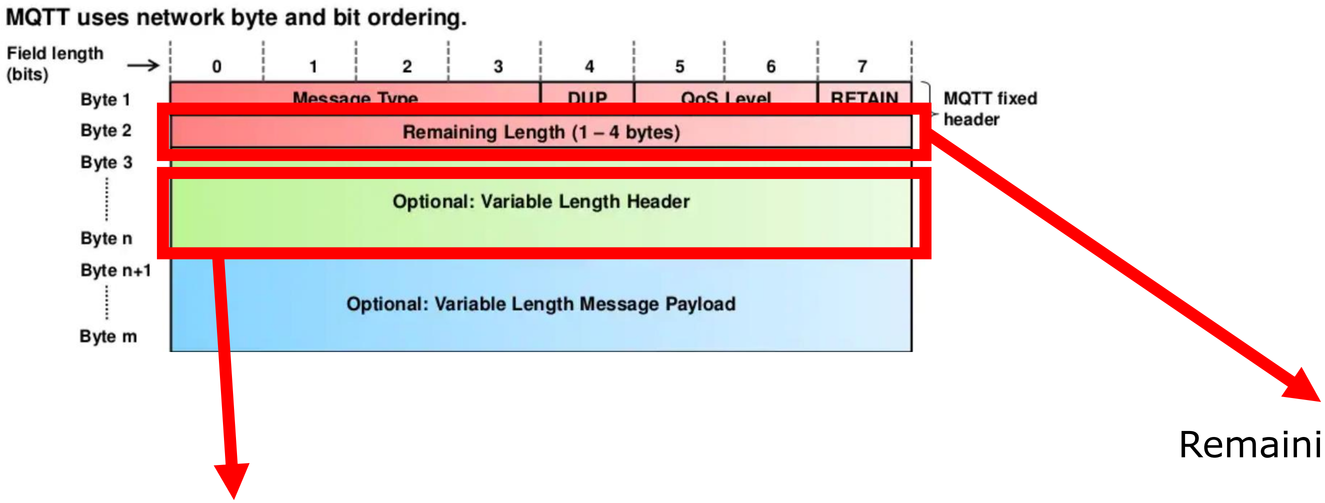
https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

Retain

- Non-Retained: If a message is published and nobody subscribes to it, then it is discarded
- The Retain flag indicates, if the message should be stored on the broker for new subscribers
- Only the last value per topic is retained

Lets try…

```
MQTT_nutshell
```



MQTT uses network byte and bit ordering.

Field length (bits) →	0	1	2	3	4	5	6	7
Byte 1	Message Type				DUP	QoS Level		RETAIN
Byte 2	Remaining Length (1 – 4 bytes)							
Byte 3 ⋮ Byte n	Optional: Variable Length Header							
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload							

Used for some MQTT Messages

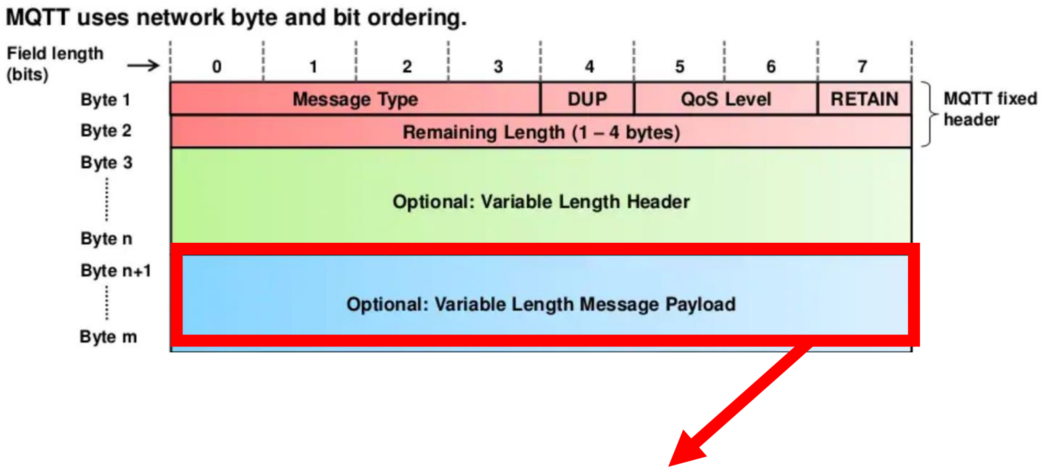
E.g. „Connect" has four parts (will not be discussed further):

- protocol name bytes,
- protocol level
- connect flags
- keepalive

Remaining Package Length

"The Remaining Length is the number of bytes left in the current packet"

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf



MQTT uses network byte and bit ordering.

Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 ⋮ Byte n	Optional: Variable Length Header								
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload								

Exemplary Content:

- Client ID: Id of the Client
- Clean Session: The session can continue across sequences of network connections: If Clean Session is set to true, then the connection is treated as a complete new one

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

Quality of Service

- You can set a QoS level for publishing and subscribing
- MQTT foresees three QoS Levels 0-2 (2 bits – see previous slide)
- QoS indicate different kind of guarantees

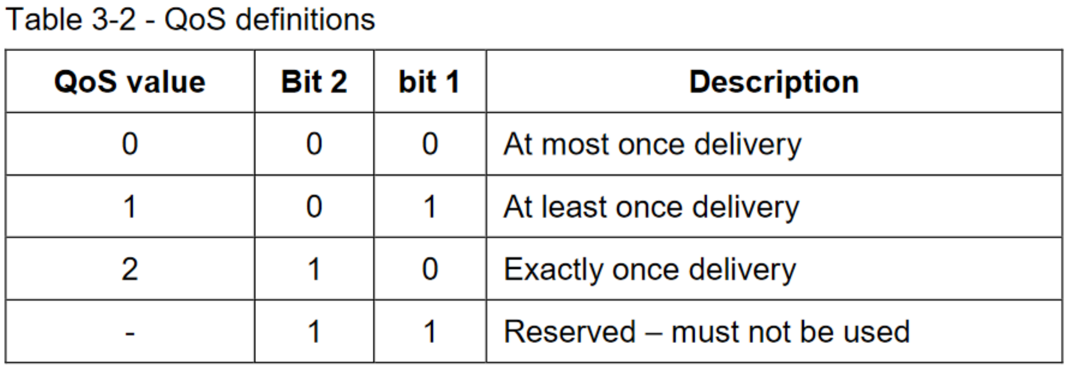


Table 3-2 - QoS definitions

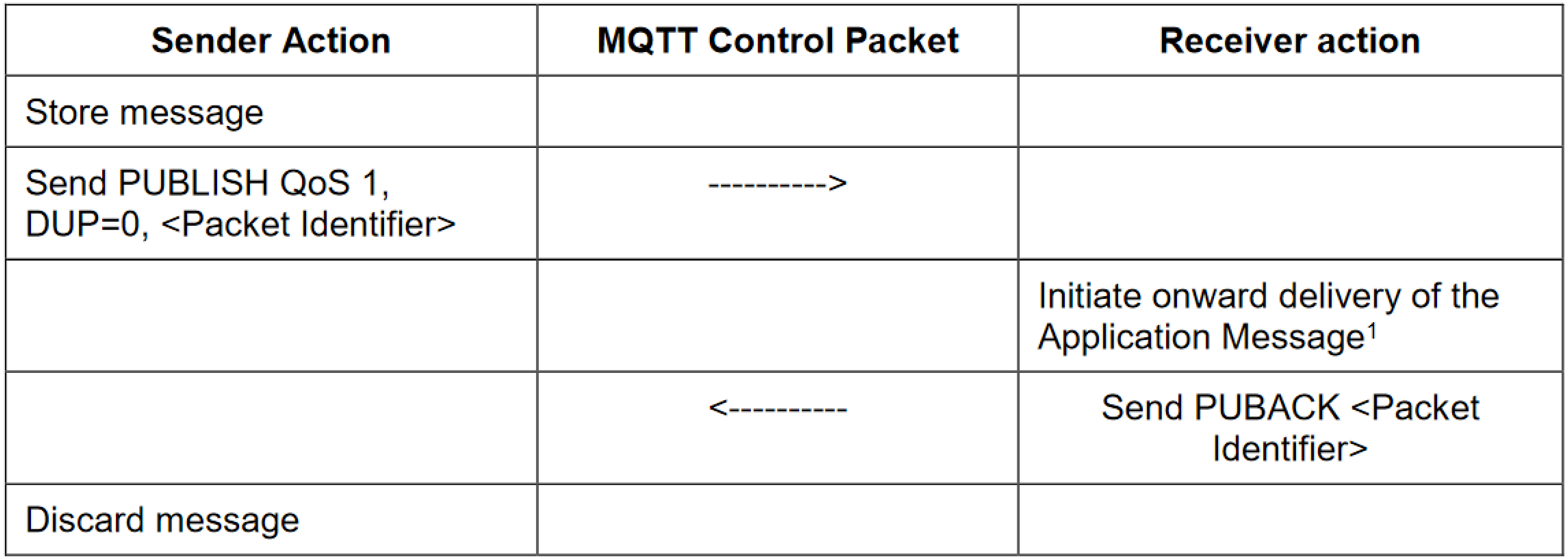
QoS value	Bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

Quality of Service

- QoS 0: The sender must send a PUBLISH packet with QoS 0 and DUP flag set to 0
  If no response arrives, the package is not re-sent: “Fire and Forget”
- QoS 1: This Quality of Service level ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH packet has a Packet Identifier in its Variable Header and is acknowledged by a PUBACK packet.
  - The Packet Identifier becomes available for reuse once the sender has received the PUBACK packet.
  - During waiting, the sender can send further packages

Figure 4.2 – QoS 1 protocol flow diagram, non-normative example



Sender Action	MQTT Control Packet	Receiver action
Store message		
Send PUBLISH QoS 1, DUP=0, <Packet Identifier>	---------->	
		Initiate onward delivery of the Application Message[1]
	<----------	Send PUBACK <Packet Identifier>
Discard message		

If a packet gets lost along the way, the sender is responsible to retransmit the message (on reconnect).

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

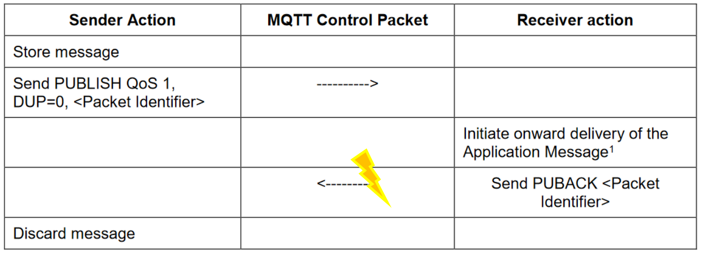
QoS 1

- 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. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

Question: Without DUP, can I distinguish between two messages?

- Package ID would indicate a duplication, but
- Package IDs can be re-used if Acknowledge was received (so unclear if new or same message)

Figure 4.2 – QoS 1 protocol flow diagram, non-normative example



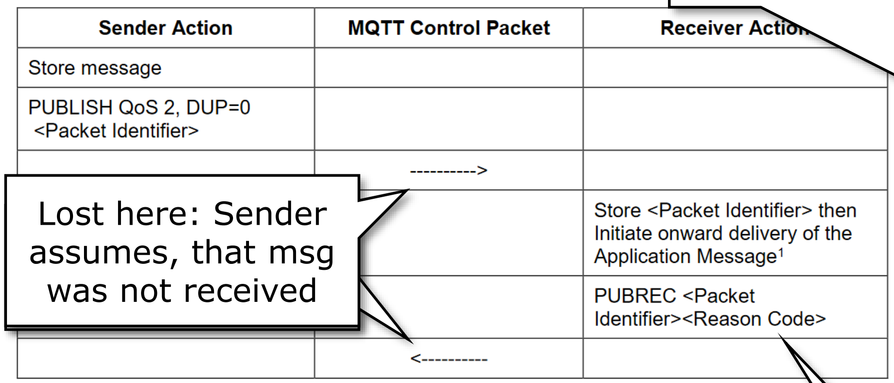
Sender Action	MQTT Control Packet	Receiver action
Store message		
Send PUBLISH QoS 1, DUP=0, <Packet Identifier>	---------->	
		Initiate onward delivery of the Application Message[1]
	<---------	Send PUBACK <Packet Identifier>
Discard message		

Quality of Service 2

PUBREL: Publish Released

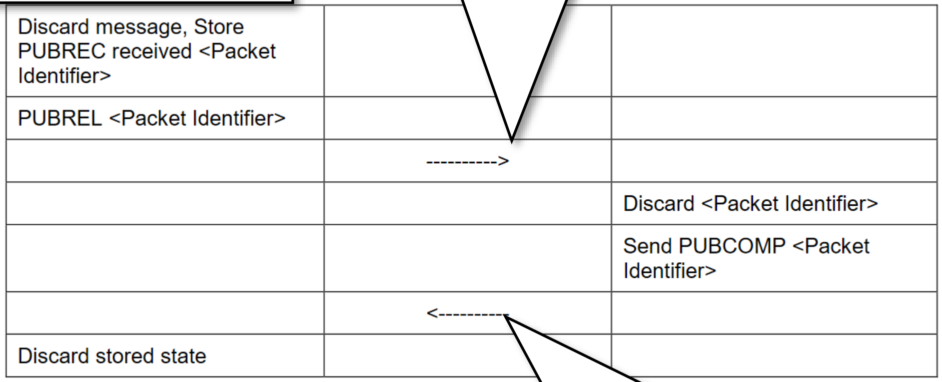
Lost here: Receiver doesn‘t know if the sender is aware that the msg arrived

Figure 4.3 – QoS 2 protocol flow diagram, non-normative example



Sender Action	MQTT Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP=0 <Packet Identifier>		
	---------->	
		Store <Packet Identifier> then Initiate onward delivery of the Application Message[1]
		PUBREC <Packet Identifier><Reason Code>
	<----------	

Lost here: Sender assumes, that msg was not received



Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	---------->	
		Discard <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<----------	
Discard stored state		

Idea: If sender doesn‘t re-send a publish msg, then it received it. Problem: Timeout, Disconnection

Lost here: Sender doesn‘t know if the receiver is aware that everything is finished

If the sender does not get a PUBREC packet from the receiver, it sends the PUBLISH packet again with a duplicate (DUP) flag until it receives an acknowledgement
If a packet gets lost along the way, the sender is responsible to retransmit the message within a reasonable amount of time.

Summary - Quality of Service

- Queuing is only possible for QoS 1+2 (offline clients)
- QoS 2 has the highest guarantee but also the highest traffic
  - Slowest protocol
- QoS 1 guarantees that message is received, however it might be received multiple times
  - If multiple messages are sent in parallel, this can cause a change of order on the broker (!)
- QoS for subscribe and publish can be different!
- Once the flow is complete, the packet identifier is available for reuse

Clean Session=false

Broker remembers the Client (using the ClientId), even if there are no subscriptions

What is stored?

- All the subscriptions of the client
- All messages with Quality of Service (QoS) of 1 or 2 flow that the client has not yet confirmed (completely)
- All new QoS 1 or 2 messages that the client missed while offline

https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages/

Retain vs Clean Session (=false)

Retain	Clean Session
...is set per topic	...is set for a client
Ensurse that the last value of a topic is deliverd to new subscribers	Ensures that all QoS 1,2 messages are deliverd to the client (with clean session false)
...is independent of the QoS	Is relevant for QoS 1,2 messages
...last retain value will be stored (even if other non-retained values follow)	All QoS>0 messages (retained as well as non-retained) will be delivered

Retained vs Persistend Session: Retained simply stores the value of a topic on the borker while clean session manages the session of a client

Excurse: Order of Delivery of messages

A Server MUST by default treat each Topic as an "Ordered Topic".

Non-Normative: The rules listed above ensure that when a stream of messages is published and subscribed to an Ordered Topic with **QoS 1**, the final copy of each message received by the subscribers **will be in the order that they were published**. If the message is re-sent the duplicate message can be received after one of the earlier messages is received. For example, a publisher might send messages in the order **1,2,3,4 but the subscriber might receive them in the order 1,2,3,2,3,4 if there is a network disconnection after message 3 has been sent.**

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

Eclipse-Mosquitto In-Flight Parameter:
The maximum number of QoS 1 and 2 messages currently inflight per client. This includes messages that are partway through handshakes and those that are being retried. Defaults to 20**. Set to 0 for no maximum. Setting to 1 will guarantee in-order delivery of QoS 1 and 2 messages.**

MQTT 5

- Request/Response Mechanism
- Topic Alias
- Automatic deletion of Topics
- Shared Subscriptions (load balancing)