MQTT

MQ TELEMETRY TRANSPORT

AN INTRODUCTION TO MQTT, A PROTOCOL FOR M2M AND IoT APPLICATIONS

Peter R. Egli
INDIGOO.COM

Contents

1. What is MQTT?

MQTT is a lightweight message queueing and transport protocol.
MQTT, as its name implies, is suited for the transport of telemetry data (sensor and actor data).

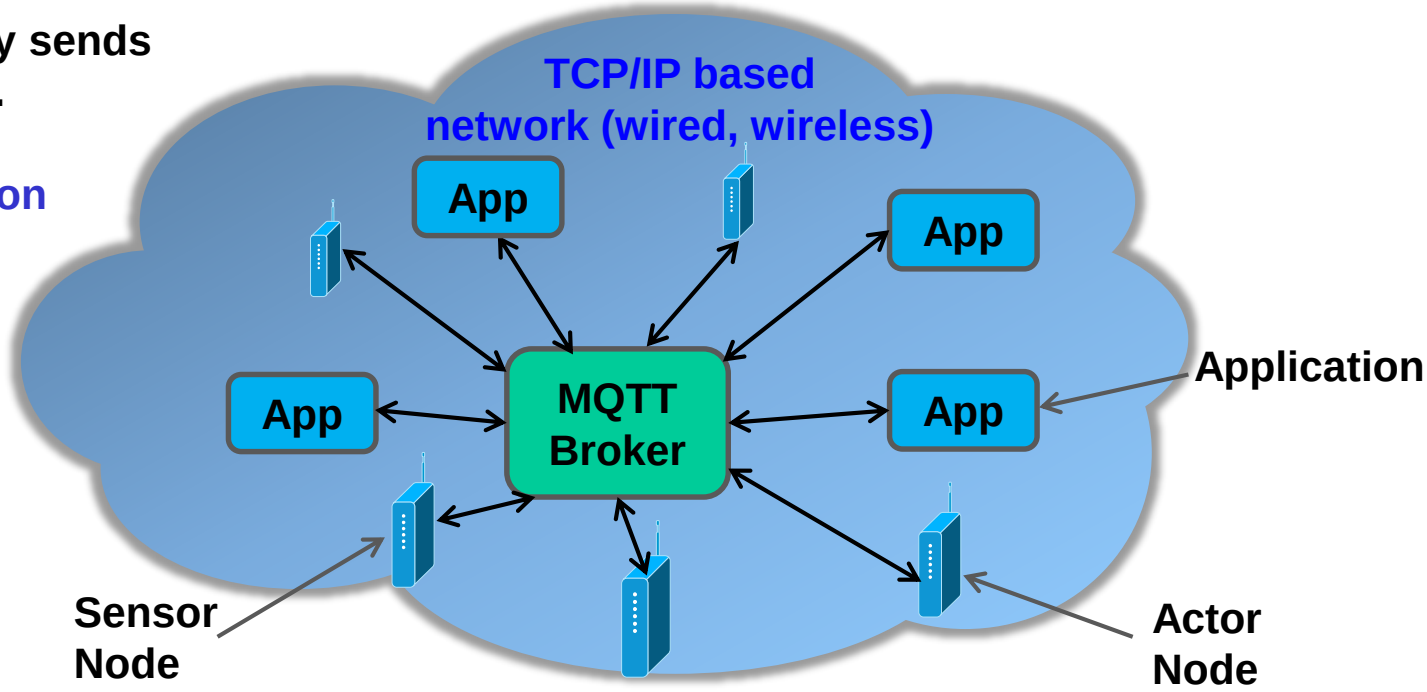
MQTT is very lightweight and thus suited for M2M (Mobile to Mobile), WSN (Wireless Sensor Networks) and ultimately IoT (Internet of Things) scenarios where sensor and actor nodes communicate with applications through the MQTT message broker.

Example:

Light sensor continuously sends sensor data to the broker.

Building control application receives sensor data from the broker and decides to activate the blinds.

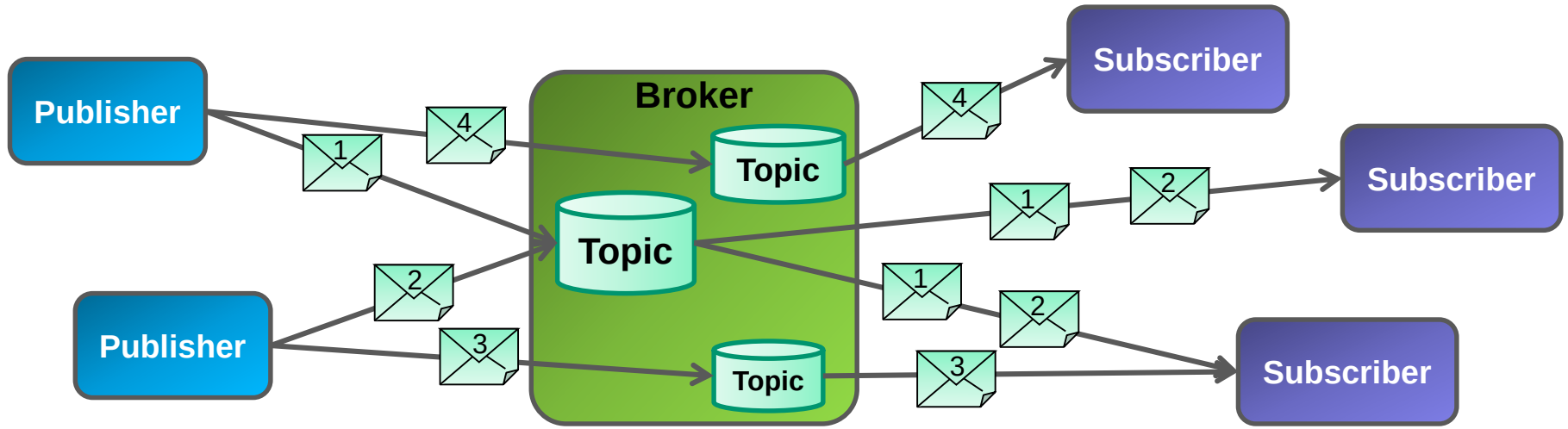
Application sends a blind activation message to the blind actor node through the broker.



2. MQTT characteristics

MQTT Key features:

- Lightweight message queueing and transport protocol
- Asynchronous communication model with messages (events)
- Low overhead (2 bytes header) for low network bandwidth applications
- Publish / Subscribe (PubSub) model
- Decoupling of data producer (publisher) and data consumer (subscriber) through topics (message queues)
- Simple protocol, aimed at low complexity, low power and low footprint implementations (e.g. WSN - Wireless Sensor Networks)
- Runs on connection-oriented transport (TCP). To be used in conjunction with 6LoWPAN (TCP header compression)
- MQTT caters for (wireless) network disruptions



3. Origins and future of MQTT standard

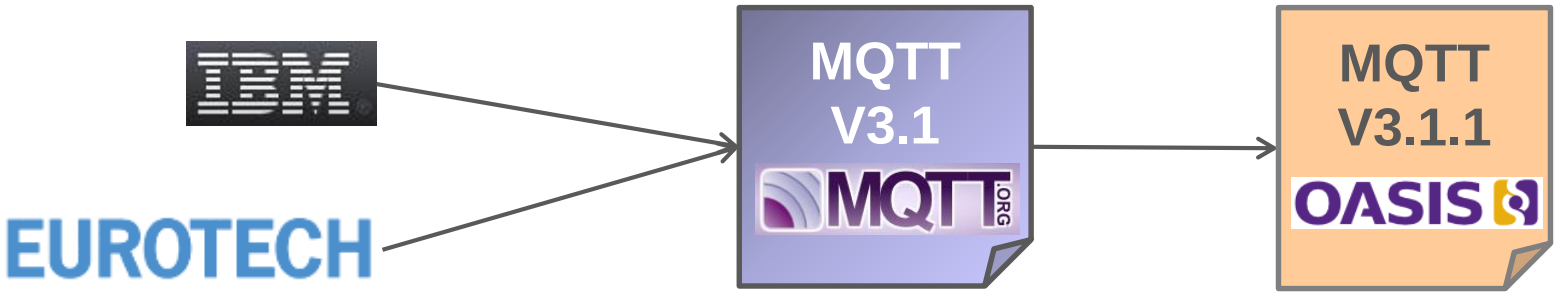
The past, present and future of MQTT:

MQTT was initially developed by IBM and Eurotech.
The previous protocol version 3.1 was made available under http://mqtt.org/.

In 2014, MQTT was adopted and published as an official standard by OASIS (published V3.1.1).
As such, OASIS has become the new home for the development of MQTT.
The OASIS TC (Technical Committee) is tasked with the further development of MQTT.

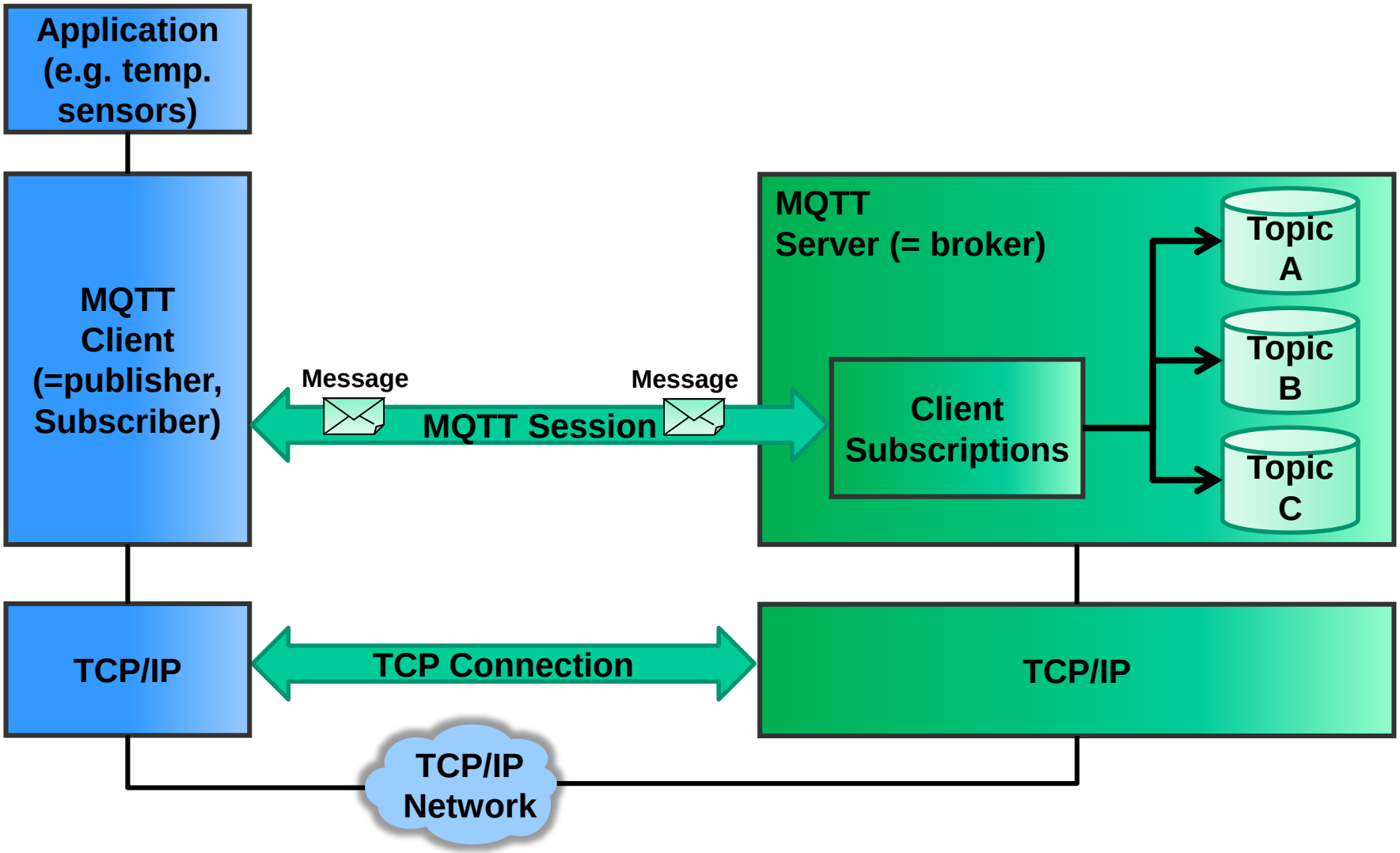
Version 3.1.1 of MQTT is backward compatible with 3.1 and brought only minor changes:

- **Changes restricted to the CONNECT message**
- **Clarification of version 3.1 (mostly editorial changes)**



4. MQTT model (1/3)

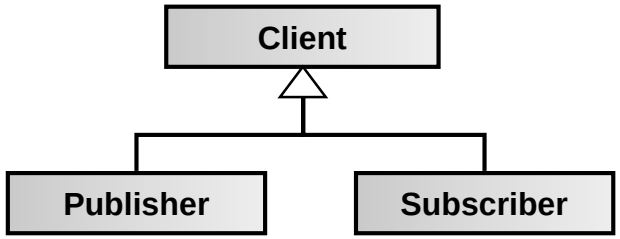
The core elements of MQTT are clients, servers (=brokers), sessions, subscriptions and topics.



4. MQTT model (2/3)

MQTT client (=publisher, subscriber):

Clients subscribe to topics to publish and receive messages.
Thus subscriber and publisher are special roles of a client.



MQTT server (=broker):

Servers run topics, i.e. receive subscriptions from clients on topics, receive messages from clients and forward these, based on client's subscriptions, to interested clients.

Topic:

Technically, topics are message queues. Topics support the publish/subscribe pattern for clients.
Logically, topics allow clients to exchange information with defined semantics.
Example topic: Temperature sensor data of a building.

4. MQTT model (3/3)

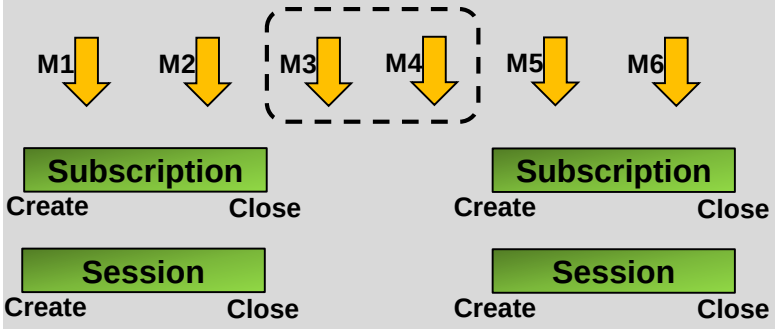
Session:

A session identifies a (possibly temporary) attachment of a client to a server. All communication between client and server takes place as part of a session.

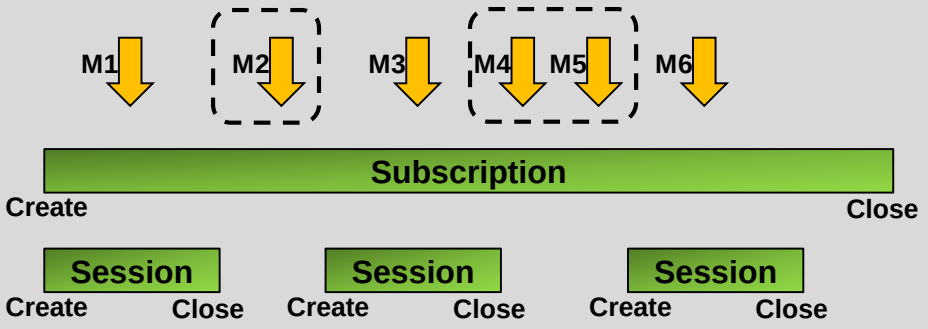
Subscription:

Unlike sessions, a subscription logically attaches a client to a topic. When subscribed to a topic, a client can exchange messages with a topic.
Subscriptions can be «transient» or «durable», depending on the clean session flag in the CONNECT message:

«Transient» subscription ends with session:
Messages M3 and M4 are not received by the client



«Durable» subscription:
Messages M2, M4 and M5 are not lost but will be received by the client as soon as it creates / opens a new session.



Message:

Messages are the units of data exchange between topic clients.
MQTT is agnostic to the internal structure of messages.

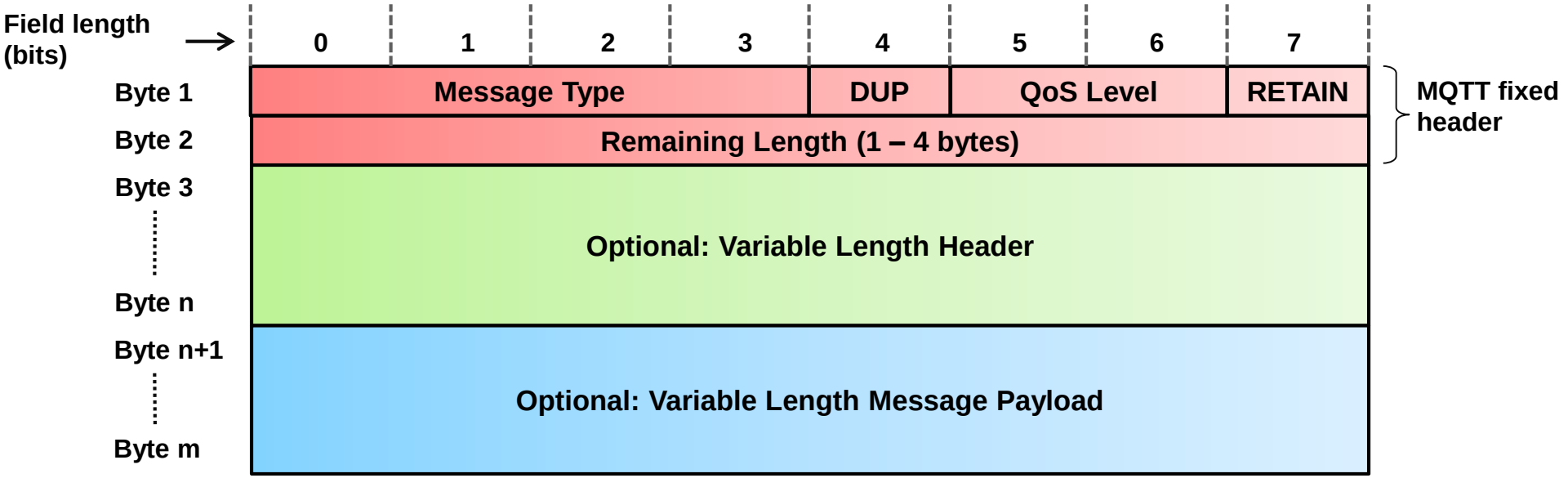
5. MQTT message format (1/14)

Message format:

MQTT messages contain a mandatory fixed-length header (2 bytes) and an optional message-specific variable length header and message payload.

Optional fields usually complicate protocol processing.
However, MQTT is optimized for bandwidth constrained and unreliable networks (typically wireless networks), so optional fields are used to reduce data transmissions as much as possible.

MQTT uses network byte and bit ordering.



5. MQTT message format (2/14)

Overview of fixed header fields:

Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	

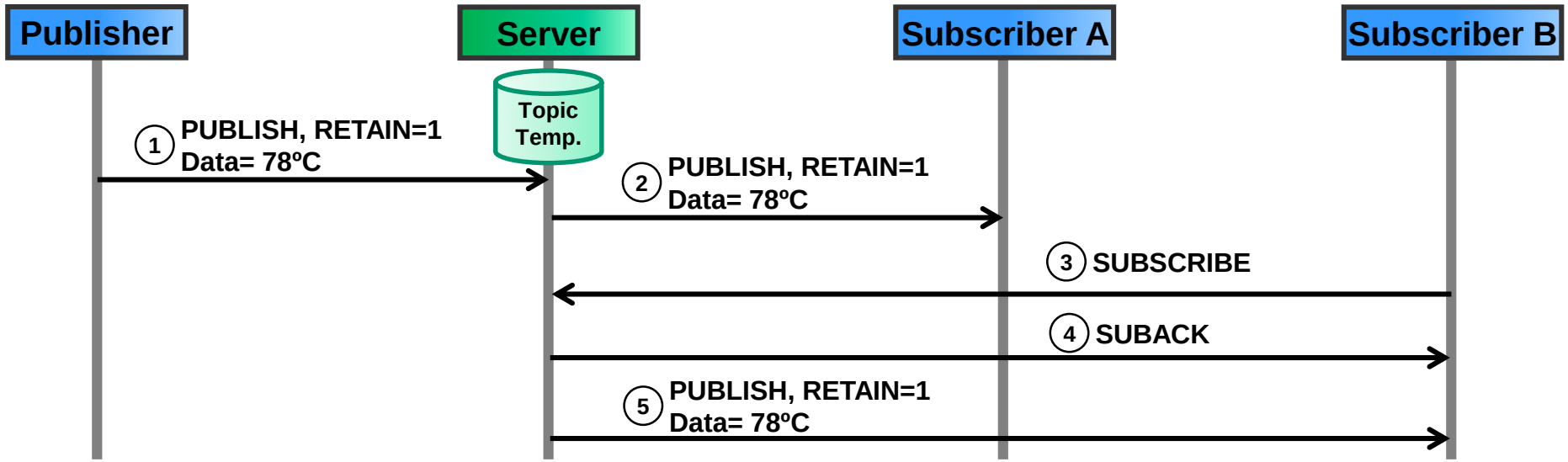
5. MQTT message format (3/14)

RETAIN (keep last message):

RETAIN=1 in a PUBLISH message instructs the server to keep the message for this topic.
When a new client subscribes to the topic, the server sends the retained message.

Typical application scenarios:
Clients publish only changes in data, so subscribers receive the last known good value.

Example:
Subscribers receive last known temperature value from the temperature data topic.
RETAIN=1 indicates to subscriber B that the message may have been published some time ago.



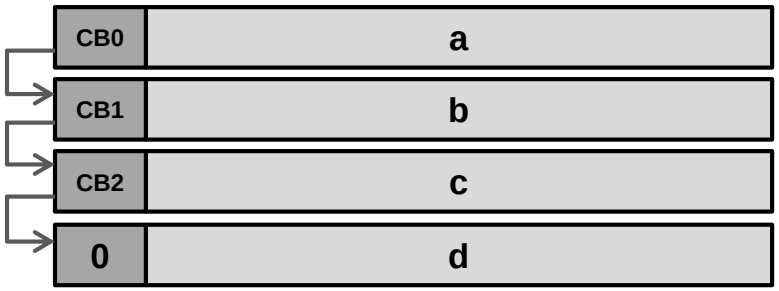
5. MQTT message format (4/14)

Remaining length (RL):

The remaining length field encodes the sum of the lengths of:
a. (Optional) variable length header
b. (Optional) payload

To save bits, remaining length is a variable length field with 1…4 bytes.
The most significant bit of a length field byte has the meaning «continuation bit» (CB). If more bytes follow, it is set to 1.

Remaining length is encoded as $a * 128^0 + b * 128^1 + c * 128^2 + d * 128^3$ and placed into the RL field bytes as follows:



Byte 0 = LSB ($a * 128^0$, CB0=1 if b > 0)

Byte 1 ($b * 128^1$, CB1=1 if c > 0)

Byte 2 ($c * 128^2$, CB2=1 if d > 0)

Byte 3 = MSB ($d * 128^3$)

Key:
LSB: Least Significant Byte
MSB: Most Significant Byte

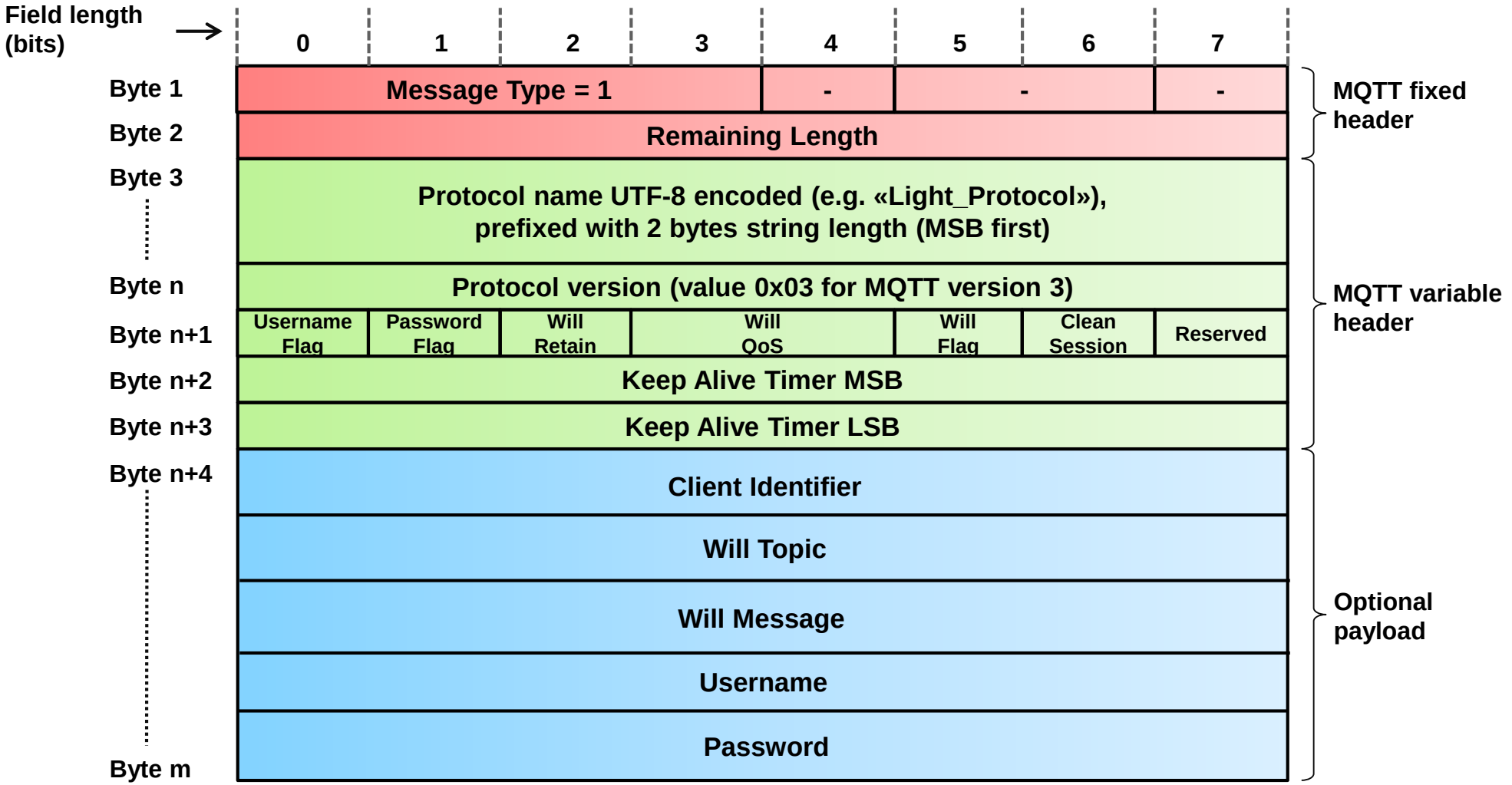
Example 1: **RL = 364 = $108*128^0+2*128^1$ → a=108, CB0=1, b=2, CB1=0, c=0, d=0, CB2=0**

Example 2: **RL = 25'897 = $41*128^0 + 74*128^1 + 1*128^2$ → a=41, CB0=1, b=74, CB1=1, c=1, CB2=0, d=0**

5. MQTT message format (5/14)

CONNECT message format:

The CONNECT message contains many session-related information as optional header fields.



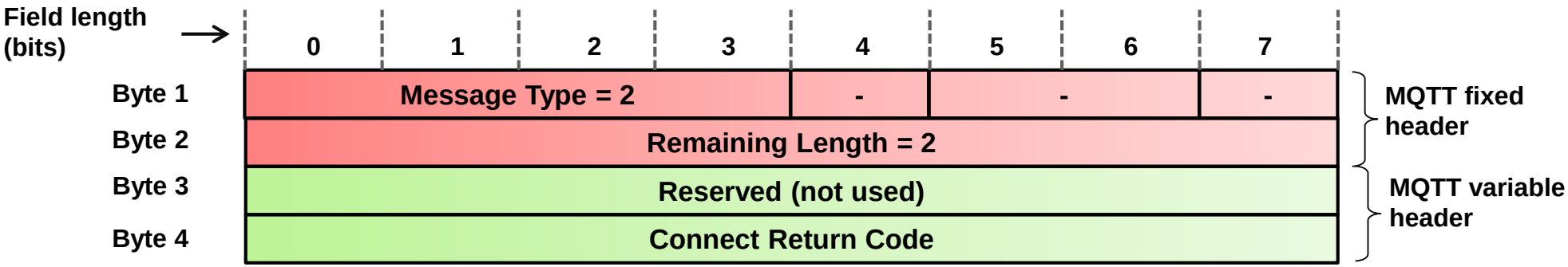
5. MQTT message format (6/14)

Overview CONNECT message fields:

CONNECT message field	Description / Values
Protocol Name	UTF-8 encoded protocol name string. Example: «Light_Protocol»
Protocol Version	Value 3 for MQTT V3.
Username Flag	If set to 1 indicates that payload contains a username.
Password Flag	If set to 1 indicates that payload contains a password. If username flag is set, password flag and password must be set as well.
Will Retain	If set to 1 indicates to server that it should retain a Will message for the client which is published in case the client disconnects unexpectedly.
Will QoS	Specifies the QoS level for a Will message.
Will Flag	Indicates that the message contains a Will message in the payload along with Will retain and Will QoS flags. More details see MQTT will message.
Clean Session	If set to 1, the server discards any previous information about the (re)-connecting client (clean new session). If set to 0, the server keeps the subscriptions of a disconnecting client including storing QoS level 1 and 2 messages for this client. When the client reconnects, the server publishes the stored messages to the client.
Keep Alive Timer	Used by the server to detect broken connections to the client. More details see Keepalive timer.
Client Identifier	The client identifier (between 1 and 23 characters)uniquely identifies the client to the server. The client identifier must be unique across all clients connecting to a server.
Will Topic	Will topic to which a will message is published if the will flag is set.
Will Message	Will message to be puslished if will flag is set.
Username and Password	Username and password if the corresponding flags are set.

5. MQTT message format (7/14)

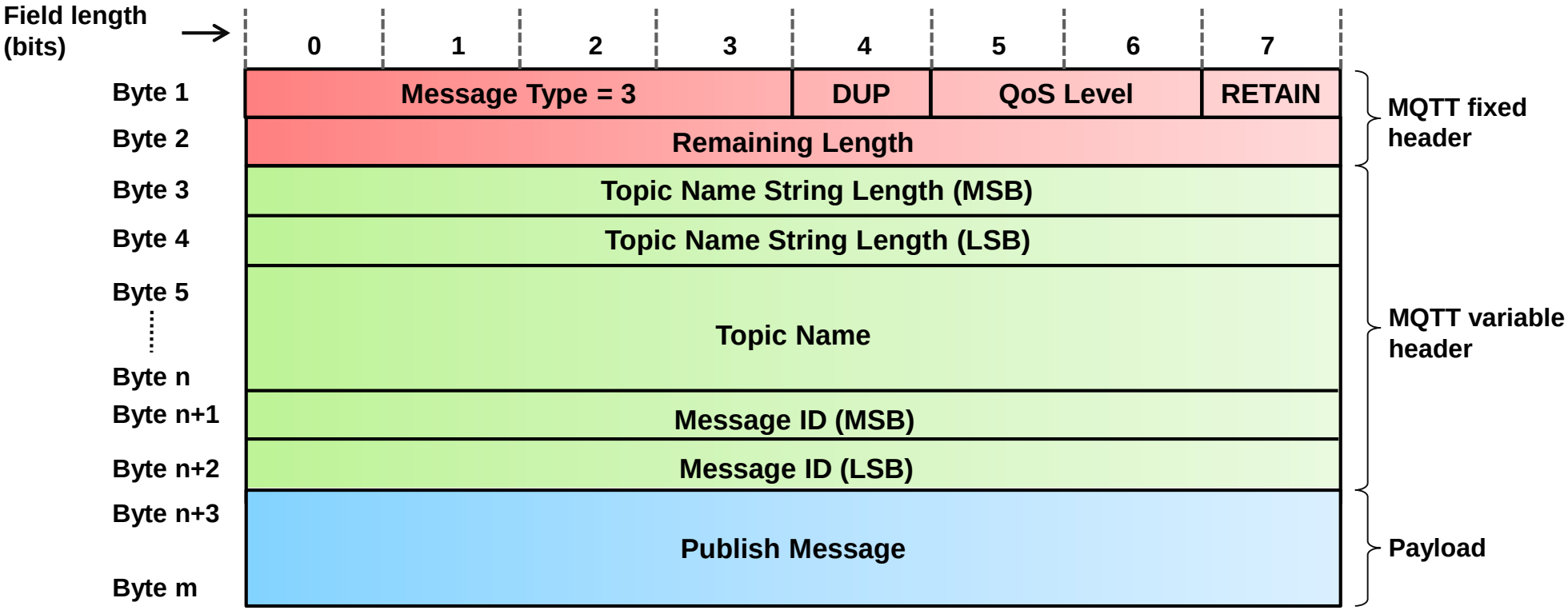
CONNACK message format:



CONNACK message field	Description / Values
Reserved	Reserved field for future use.
Connect Return Code	0: Connection Accepted 1: Connection Refused, reason = unacceptable protocol version 2: Connection Refused, reason = identifier rejected 3: Connection Refused, reason = server unavailable 4: Connection Refused, reason = bad user name or password 5: Connection Refused, reason = not authorized 6-255: Reserved for future use

5. MQTT message format (8/14)

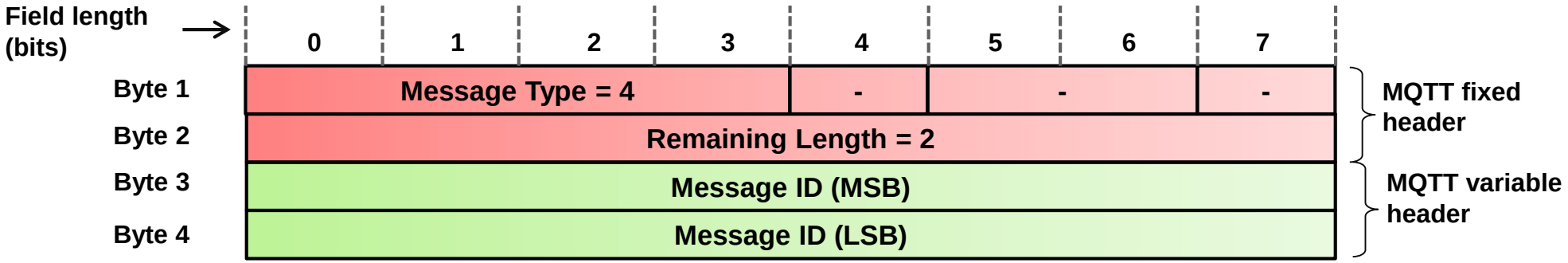
PUBLISH message format:



PUBLISH message field	Description / Values
Topic Name with Topic Name String Length	Name of topic to which the message is published. The first 2 bytes of the topic name field indicate the topic name string length.
Message ID	A message ID is present if QoS is 1 (At-least-once delivery, acknowledged delivery) or 2 (Exactly-once delivery).
Publish Message	Message as an array of bytes. The structure of the publish message is application-specific.

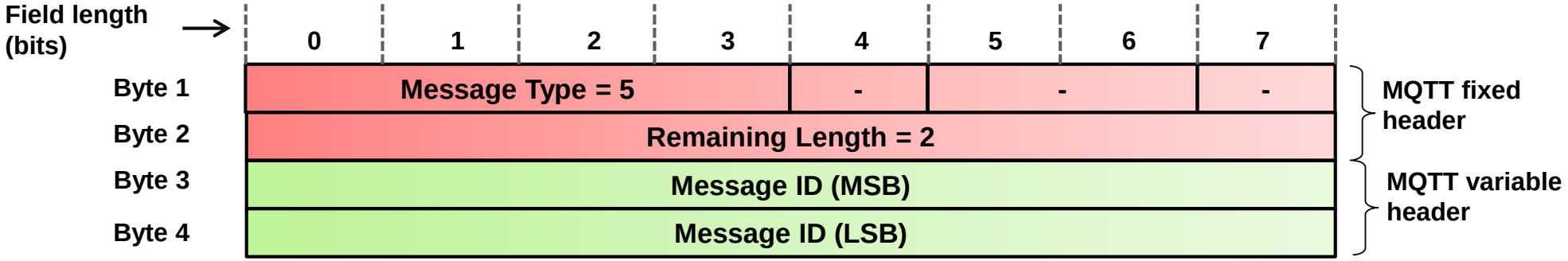
5. MQTT message format (9/14)

PUBACK message format:



PUBACK message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

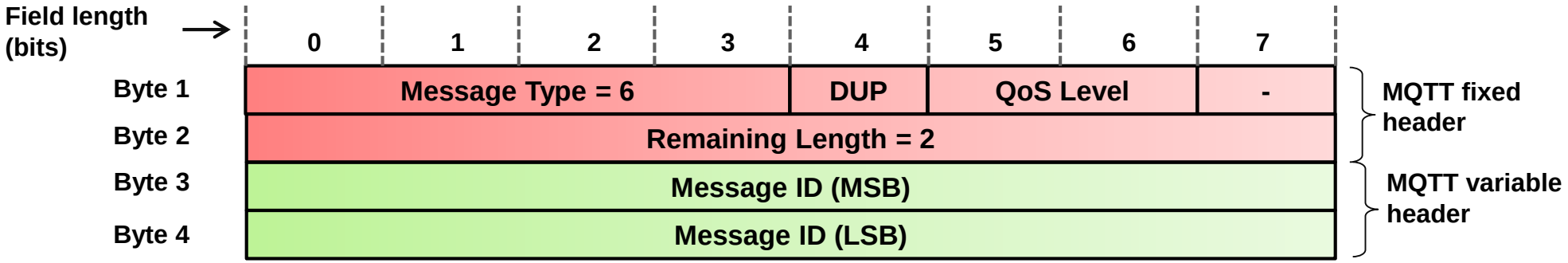
PUBREC message format:



PUBREC message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

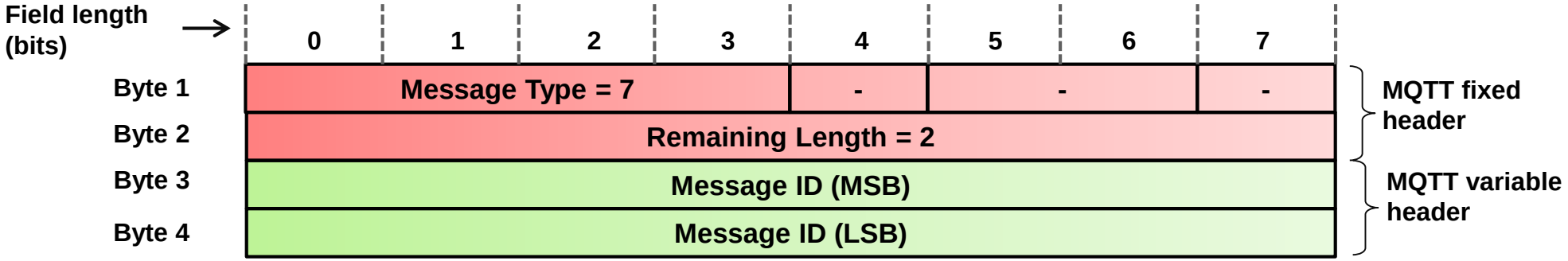
5. MQTT message format (10/14)

PUBREL message format:



PUBREL message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

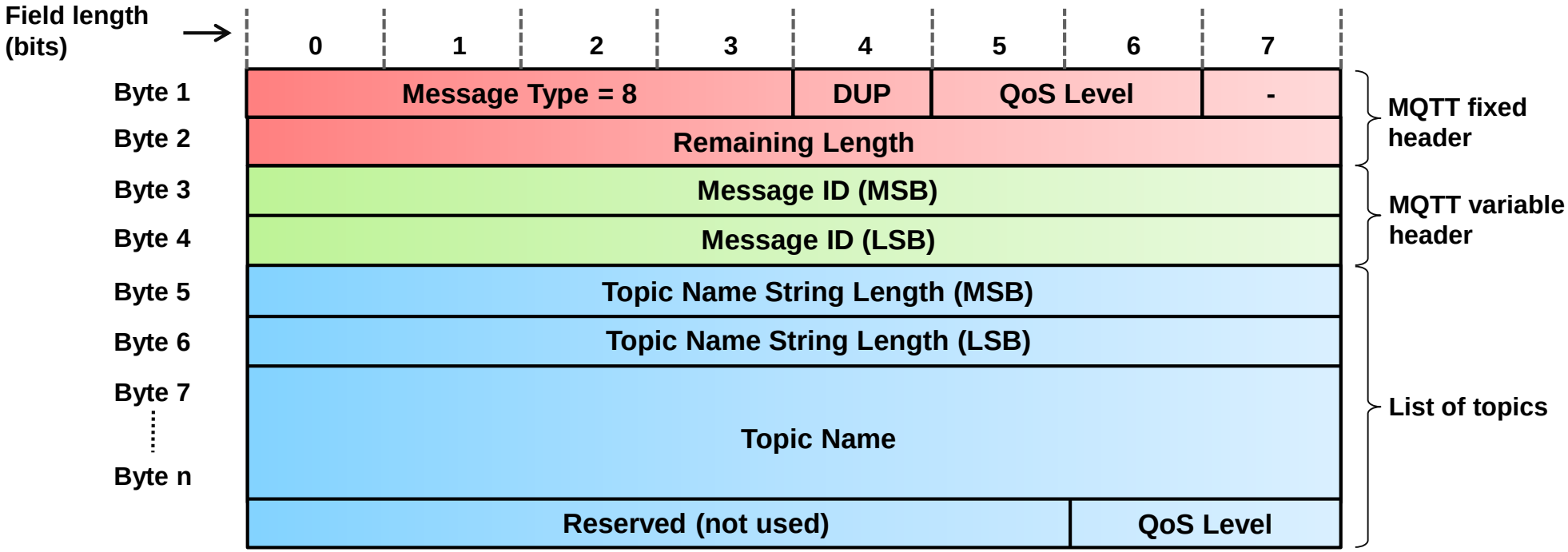
PUBCOMP message format:



PUBCOMP message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

5. MQTT message format (11/14)

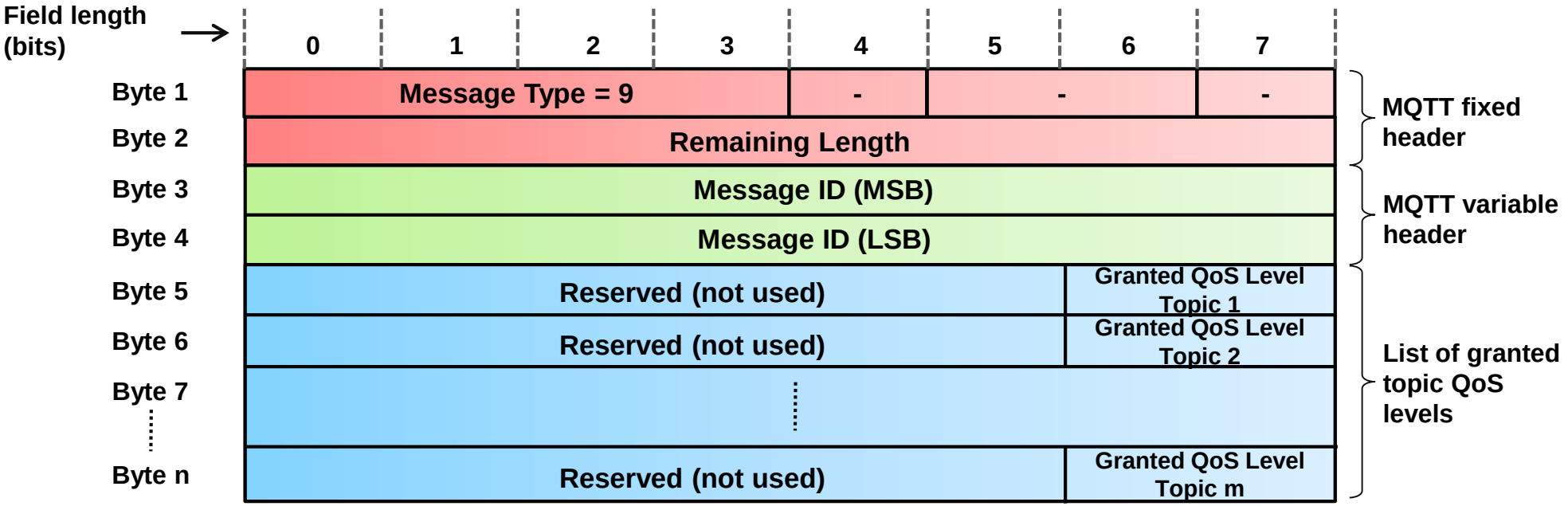
SUBSCRIBE message format:



SUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the SUBSCRIBE message since these have a QoS level of 1.
Topic Name with Topic Name String Length	Name of topic to which the client subscribes. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names along with their requested QoS level may appear in a SUBSCRIBE message.
QoS Level	QoS level at which the clients wants to receive messages from the given topic.

5. MQTT message format (12/14)

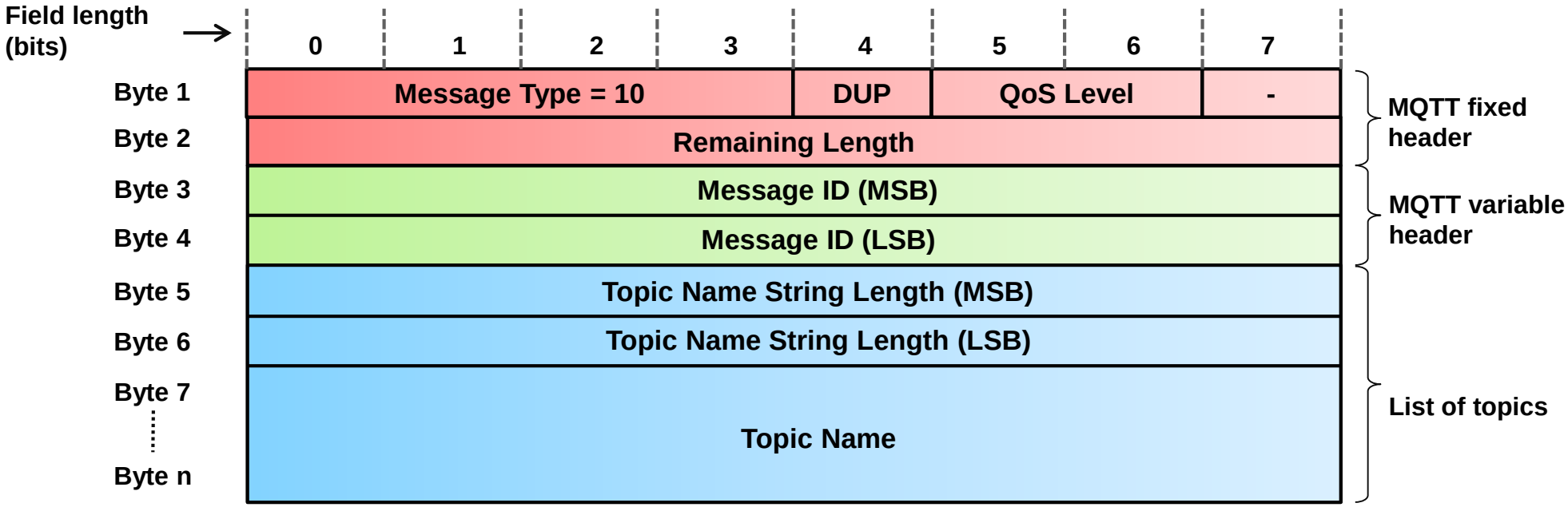
SUBACK message format:



SUBACK message field	Description / Values
Message ID	Message ID of the SUBSCRIBE message to be acknowledged.
Granted QoS Level for Topic	List of granted QoS levels for the topics list from the SUBSCRIBE message.

5. MQTT message format (13/14)

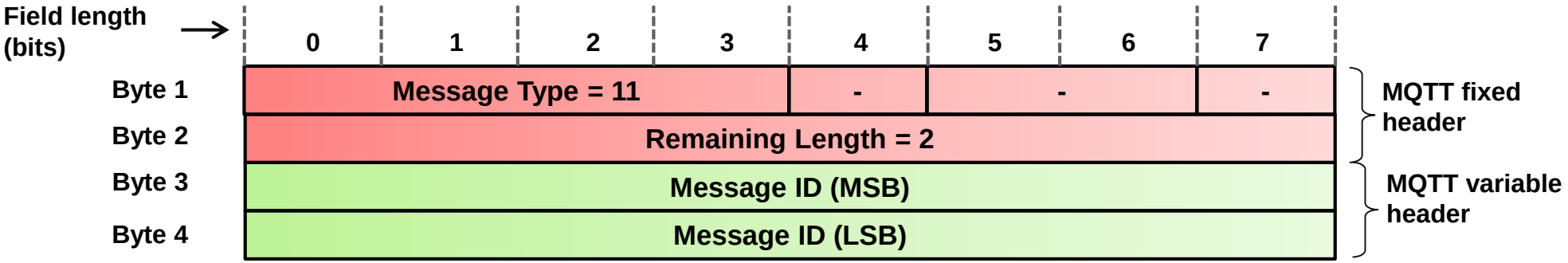
UNSUBSCRIBE message format:



UNSUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the UNSUBSCRIBE message (UNSUBSCRIBE messages have a QoS level of 1).
Topic Name with Topic Name String Length	Name of topic from which the client wants to unsubscribe. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names may appear in an UNSUBSCRIBE message.

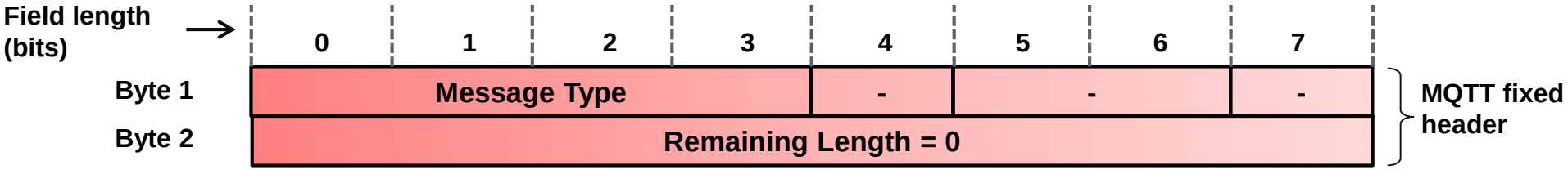
5. MQTT message format (14/14)

UNSUBACK message format:



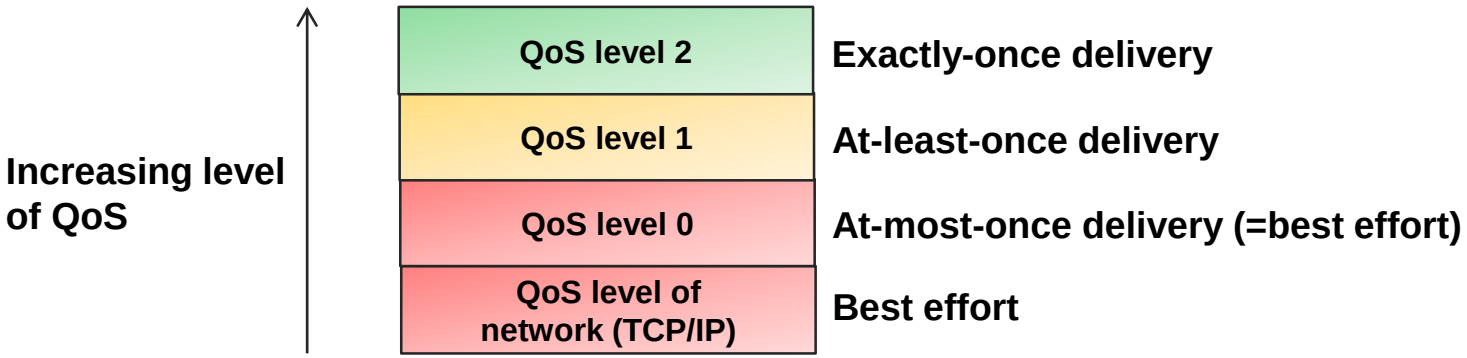
UNSUBACK message field	Description / Values
Message ID	The message ID of the UNSUBSCRIBE message to be acknowledged.

DISCONNECT, PINGREQ, PINGRESP message formats:



6. MQTT QoS (1/2)

MQTT provides the typical delivery quality of service (QoS) levels of message oriented middleware.
Even though TCP/IP provides guaranteed data delivery, data loss can still occur if a TCP connection breaks down and messages in transit are lost.
Therefore MQTT adds 3 quality of service levels on top of TCP.



QoS level 0:

At-most-once delivery («best effort»).
Messages are delivered according to the delivery guarantees of the underlying network (TCP/IP).
Example application: Temperature sensor data which is regularly published. Loss of an individual value is not critical since applications (consumers of the data) will anyway integrate the values over time and loss of individual samples is not relevant.

6. MQTT QoS (2/2)

QoS level 1:

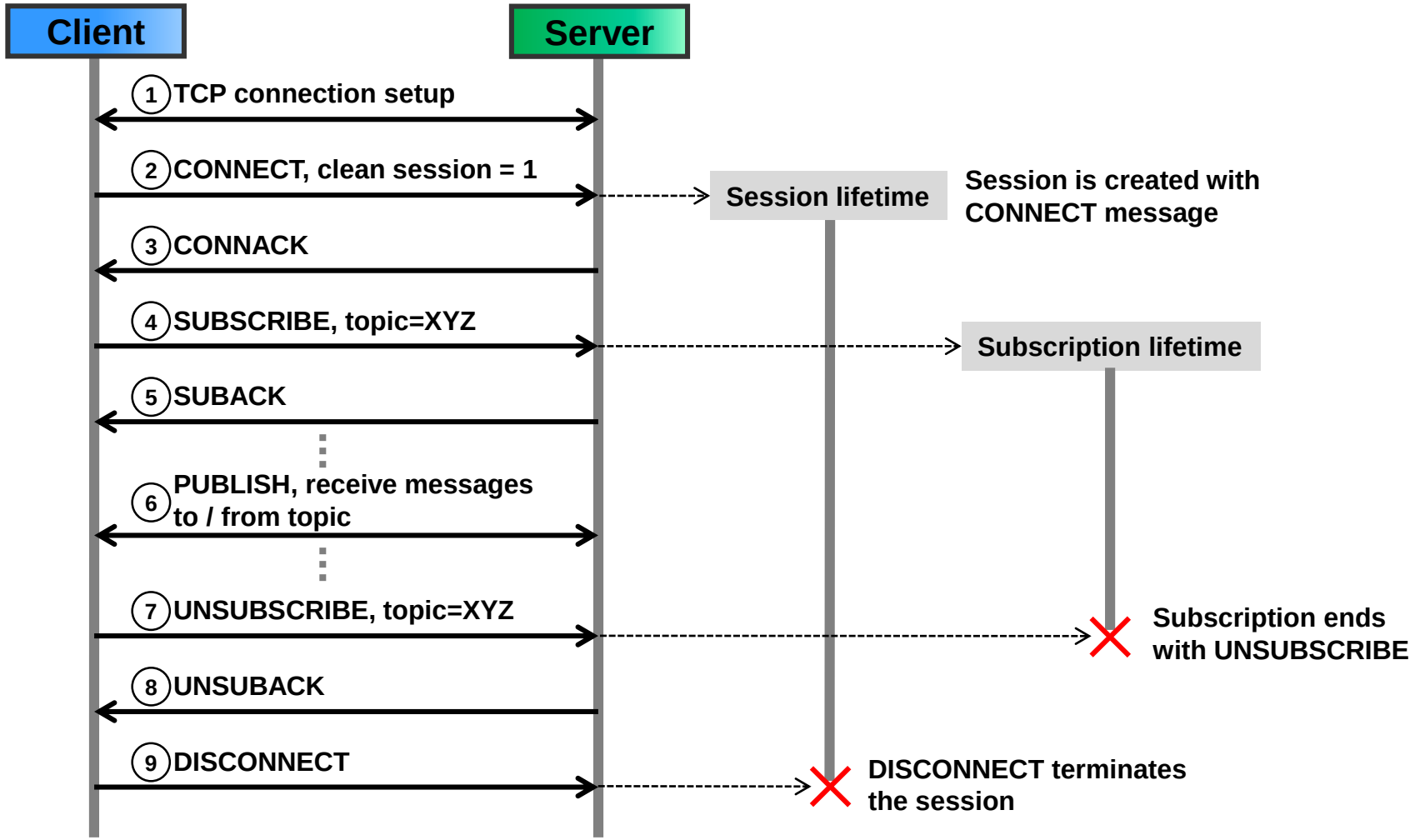
At-lest-once delivery. Messages are guaranteed to arrive, but there may be duplicates.
Example application: A door sensor senses the door state. It is important that door state changes (closed→open, open→closed) are published losslessly to subscribers (e.g. alarming function). Applications simply discard duplicate messages by evaluating the message ID field.

QoS level 2:

Exactly-once delivery.
This is the highest level that also incurs most overhead in terms of control messages and the need for locally storing the messages.
Exactly-once is a combination of at-least-once and at-most-once delivery guarantee.
Example application: Applications where duplicate events could lead to incorrect actions, e.g. sounding an alarm as a reaction to an event received by a message.

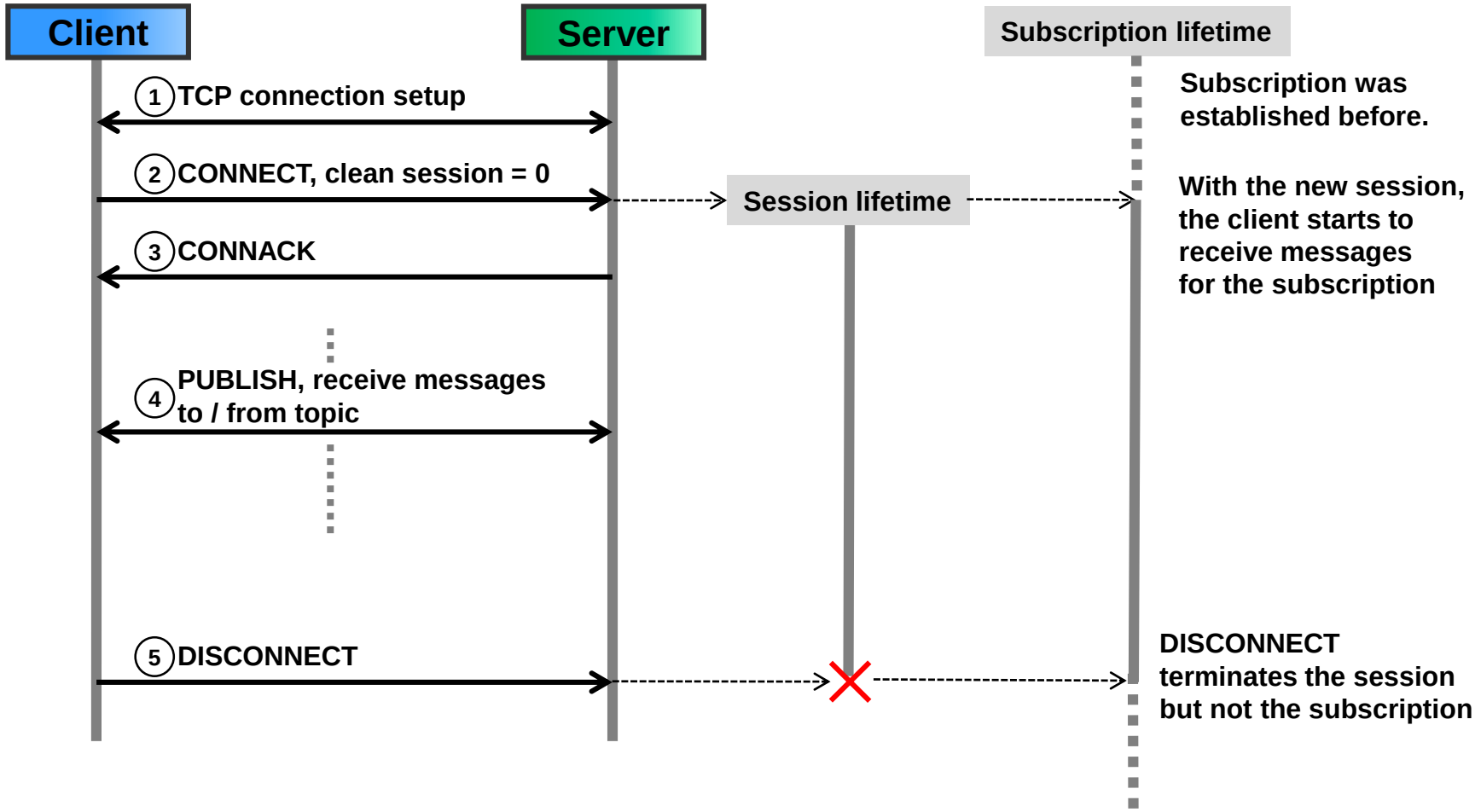
7. CONNECT and SUBSCRIBE message sequence (1/2)

Case 1: Session and subscription setup with clean session flag = 1 («transient» subscription)



7. CONNECT and SUBSCRIBE message sequence (2/2)

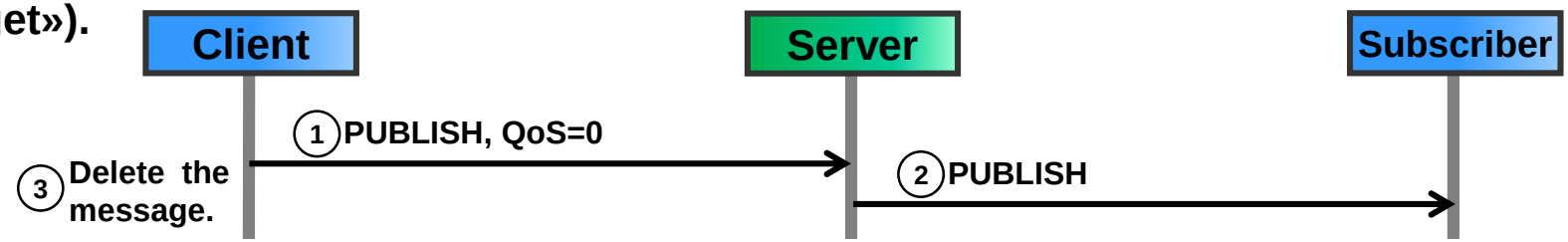
Case 2: Session and subscription setup with clean session flag = 0 («durable» subscription)



8. PUBLISH message flows (1/2)

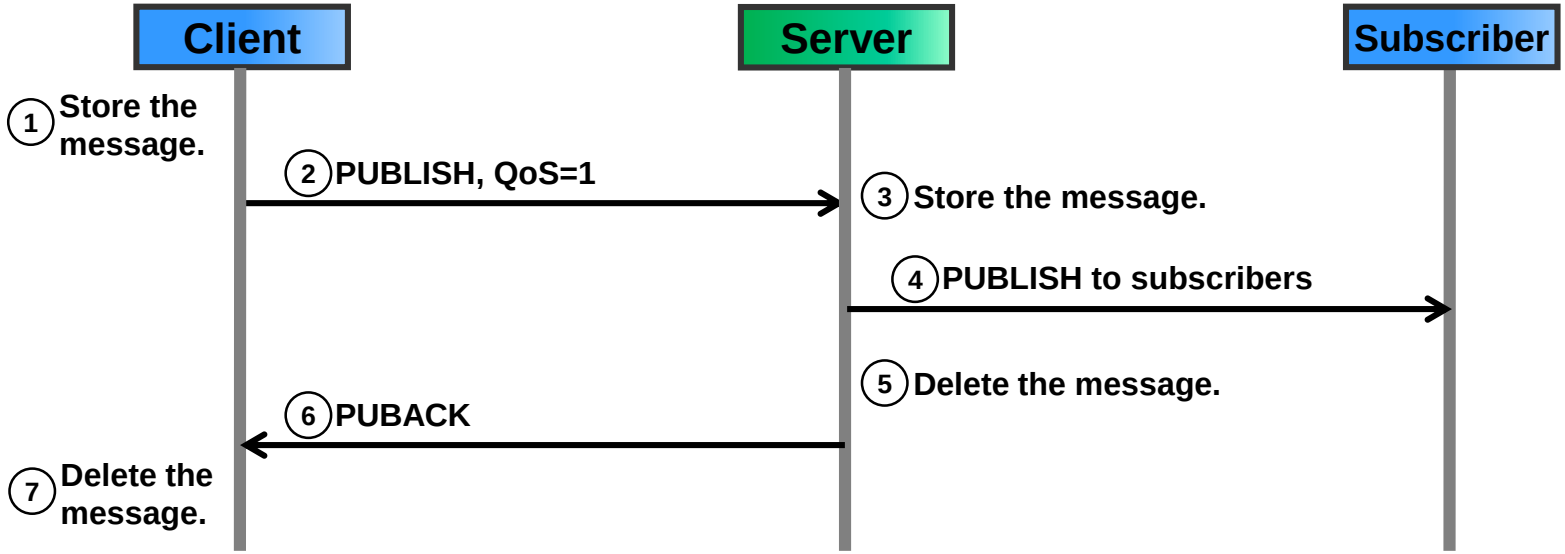
QoS level 0:

With QoS level 0, a message is delivered with at-most-once delivery semantics («fire-and-forget»).



QoS level 1:

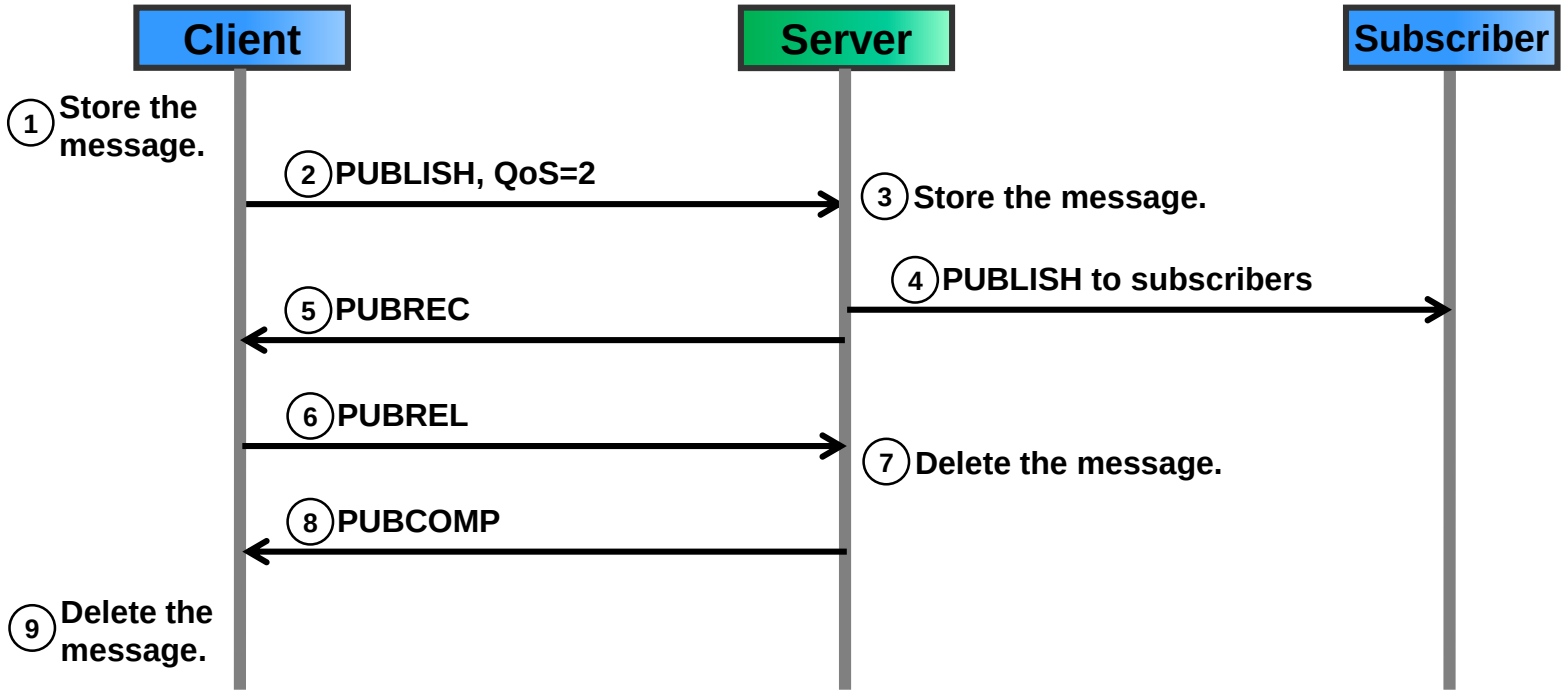
QoS level 1 affords at-least-once delivery semantics. If the client does not receive the PUBACK in time, it re-sends the message.



8. PUBLISH message flows (2/2)

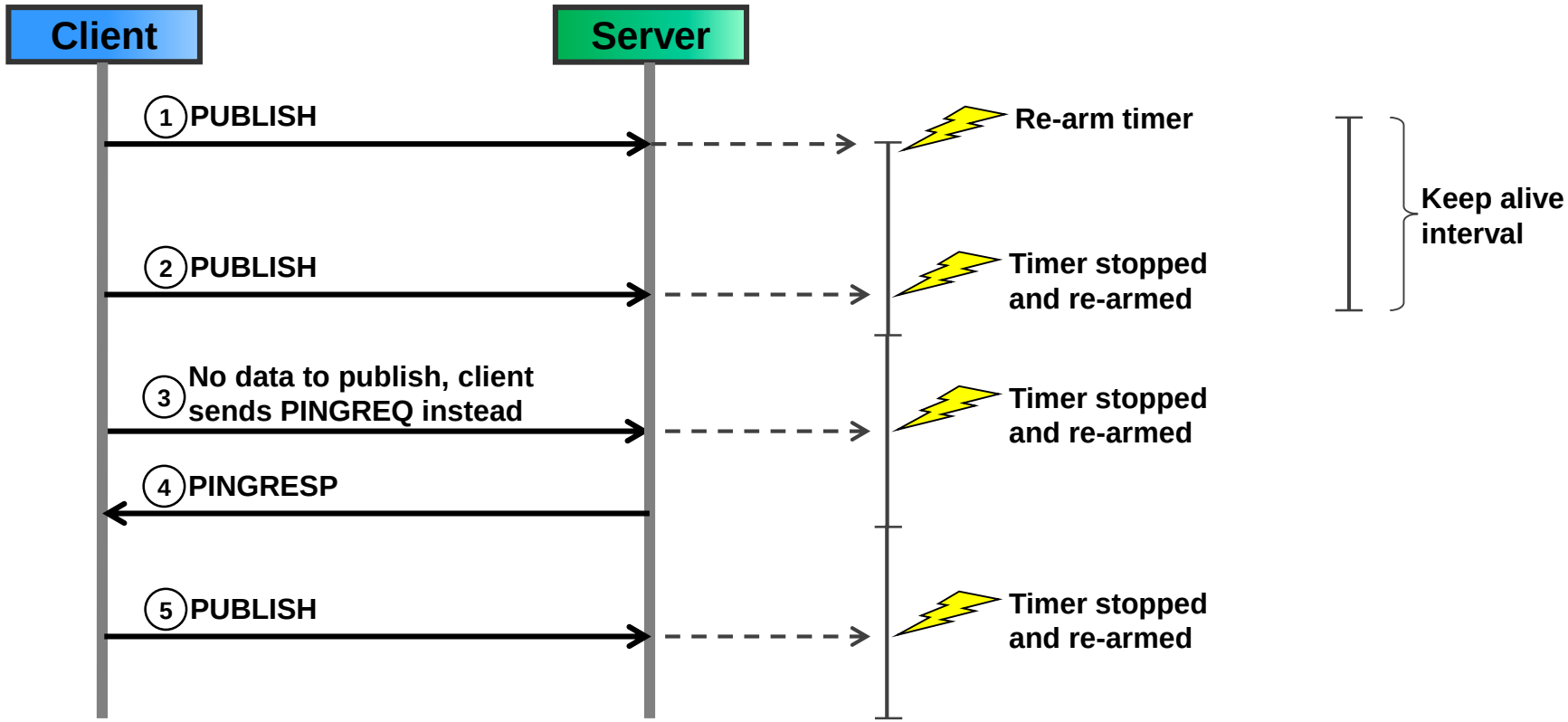
QoS level 2:

QoS level 2 affords the highest quality delivery semantics exactly-once, but comes with the cost of additional control messages.



9. Keep alive timer, breath of live with PINGREQ

The keep alive timer defines the maximum allowable time interval between client messages.
The timer is used by the server to check client's connection status.
After 1.5 * keepalive-time is elapsed, the server disconnects the client (client is granted a grace period of an additional 0.5 keepalive-time).
In the absence of data to be sent, the client sends a PINGREQ message instead.
Typical value for keepalive timer are a couple of minutes.



10. MQTT will message

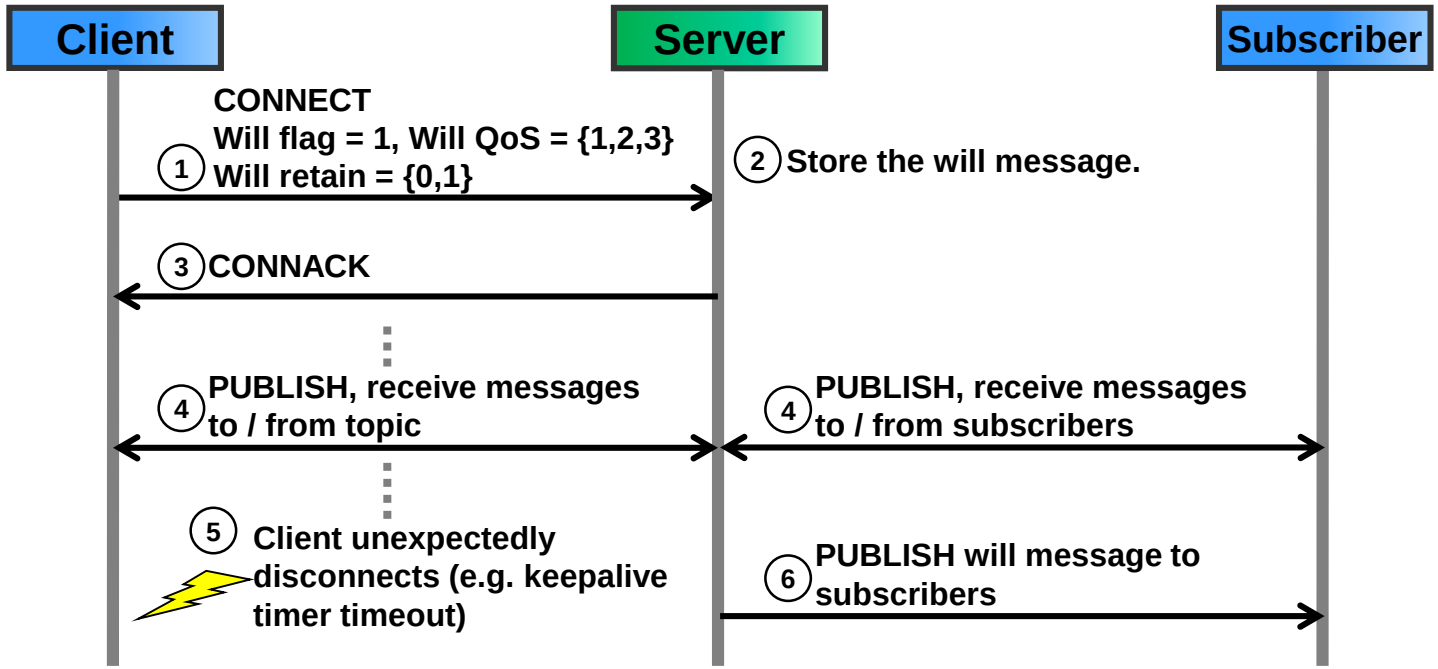
Problem:

In case of an unexpected client disconnect, depending applications (subscribers) do not receive any notification of the client's demise.

MQTT solution:

Client can specify a will message along with a will QoS and will retain flag in the CONNECT message payload.

If the client unexpectedly disconnects, the server sends the will message on behalf of the client to all subscribers («last will»).



11. Topic wildcards

Problem:

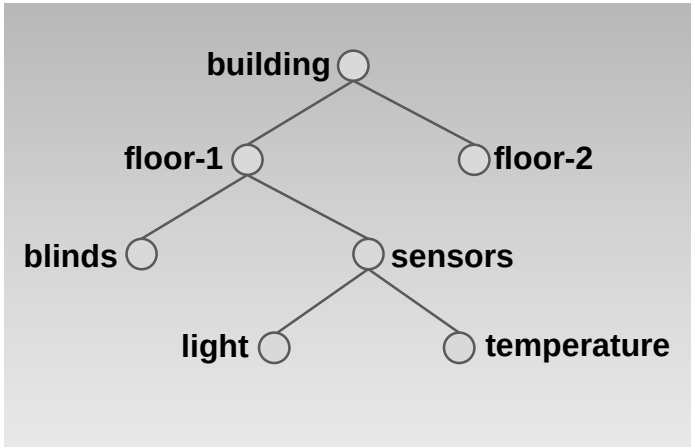
Subscribers are often interested in a great number of topics.
Individually subscribing to each named topic is time- and resource-consuming.

MQTT solution:

Topics can be hierarchically organized through wildcards with path-type topic strings and the wildcard characters '+' and '#'.
Subscribers can subscribe for an entire sub-tree of topics thus receiving messages published to any of the sub-tree's nodes.

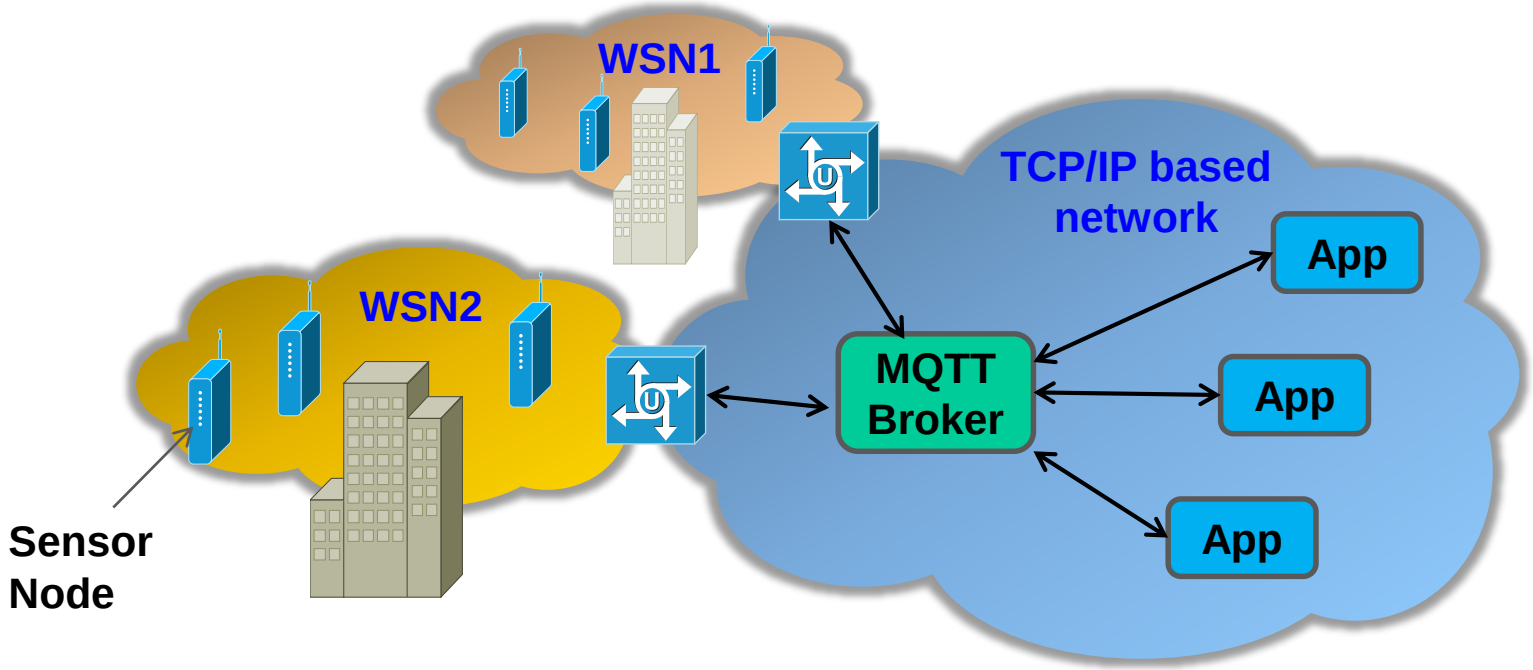
Topic string special character	Description
/	Topic level separator. Example: *building / floor-1 / sensors / temperature*
+	Single level wildcard. Matches one topic level. Examples: *building / floor-1 / +* (matches *building / floor-1 / blinds* and *building / floor-1 / sensors*) *building / + / sensors* (matches *building / floor-1 / sensors* and *building / floor-2 / sensors*)
#	Multi level wildcard. Matches multiple topic levels. Examples: *building / floor-1 / #* (matches all nodes under *building / floor-1*) *building / # / sensors* (invalid, '#' must be last character in topic string)

Example topic tree:



12. MQTT-S (1/2)

WSNs (Wireless Sensor Networks) usually do not have TCP/IP as transport layer. They have their own protocol stack such as ZigBee on top of IEEE 802.15.4 MAC layer. Thus, MQTT which is based on TCP/IP cannot be directly run on WSNs.
WSNs are connected to traditional TCP/IP networks through gateway devices.



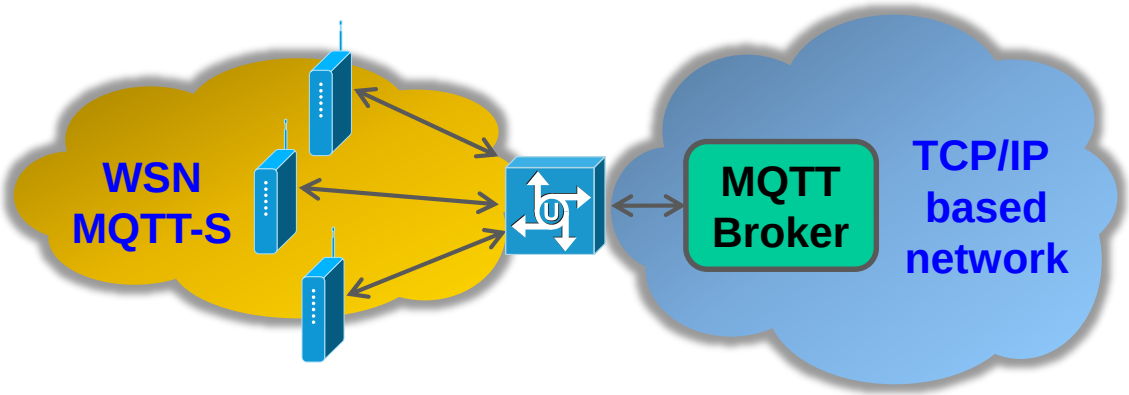
MQTT-S is an extension of MQTT for WSNs.
MQTT-S is aimed at constrained low-end devices, usually running on a batttery, such as ZigBee devices.

12. MQTT-S (2/2)

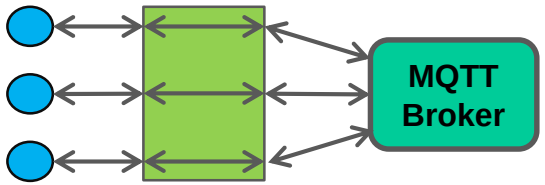
MQTT-S is a largely based on MQTT, but implements some important optimizations for wireless networks:

- **Topic string replaced by a topic ID (fewer bytes necessary)**
- **Predefined topic IDs that do not require a registration**
- **Discovery procedure for clients to find brokers (no need to statically configure broker addresses)**
- **Persistent will message (in addition to persistent subscriptions)**
- **Off-line keepalive supporting sleeping clients (will receive buffered messages from the server once they wake up)**

MQTT-S gateways (transparent or aggregating) connect MQTT-S domains (WSNs) with MQTT domains (traditional TCP/IP based networks).



Transparent gateway:
→ 1 connection to broker per client



Aggregating gateway:
→ only 1 connection to the broker