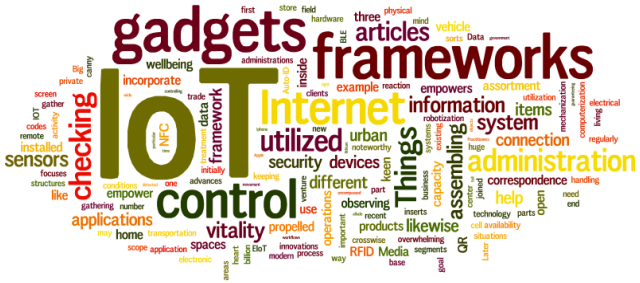
CS578: Internet of Things

MQTT:
Message Queuing Telemetry Transport



Dr. Manas Khatua

Assistant Professor

Dept. of CSE, IIT Guwahati

E-mail: manaskhatua@iitg.ac.in

What is MQTT?

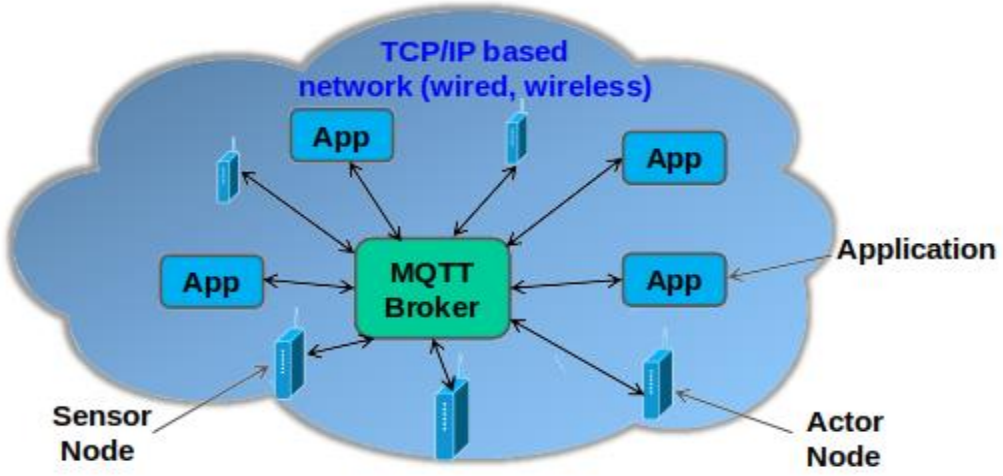
- MQTT is message queueing and transport protocol.
- **Lightweight protocol**
- Suited for the **transport of telemetry data** (sensor and actor data), and
- thus for **M2M** (Mobile to Mobile), **WSN** (Wireless Sensor Networks) application

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**.

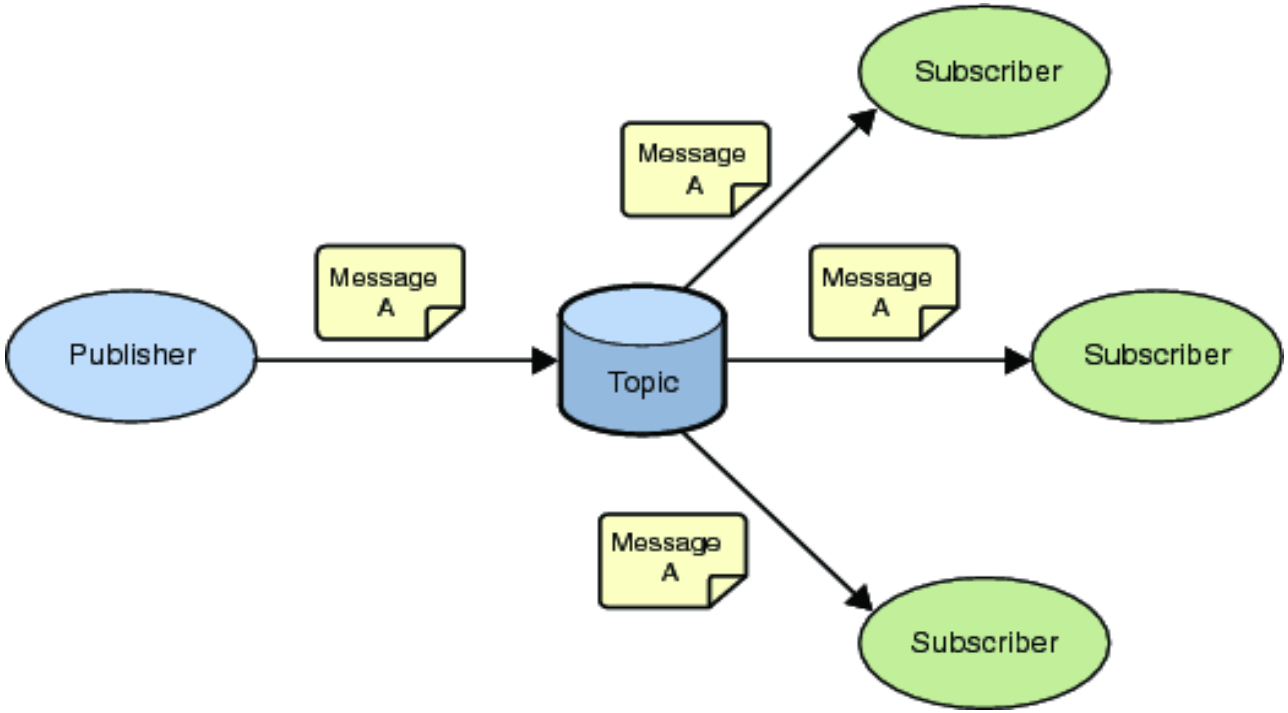
- Invented by Dr. Andy Stanford-Clark of IBM and Arlen Nipper of Arcom (now Eurotech) in 1999

- Used by
  - Amazon Web Services (AWS),
  - IBM WebSphere MQ,
  - Microsoft Azure IoT,
  - Adafruit,
  - Facebook Messenger,
  - etc.



MQTT Characteristics

- **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)
- Runs on connection-oriented transport (**TCP**). To be used in conjunction with **6LoWPAN** (TCP header compression)
- MQTT caters for (wireless) **network disruptions**



Publish Subscribe Messaging

Terminology

- A producer sends (**publishes**) a message (**publication**) on a topic (**subject**)
- A consumer **subscribes** (makes a subscription) for messages on a topic (**subject**)
- A message **server / broker** matches publications to subscriptions

Who will get the message ?

- If **no matches** the message is **discarded**
- If **one or more matches** the message is delivered to **each** matching subscriber/consumer

Topic

- A topic forms the namespace is hierarchical with each **"sub topic"** separated by a **/**
- An **example** topic space :
  - **A house publishes information about itself on:**
    <country>/<region>/<town>/<postcode>/<house>/energyConsumption
    <country>/<region>/<town>/<postcode>/<house>/solarEnergy
  - **And subscribes for control commands:**
    <country>/<region>/<town>/<postcode>/<house>/thermostat/setTemp

Publish Subscribe Messaging

Wildcards

- A subscriber can subscribe to an absolute topic or can use **wildcards**:
  - Single-level wildcards **"+"** can appear anywhere in the topic string

  For example:
  - Energy consumption for 1 house in Hursley
    - *UK/Hants/Hursley/SO212JN/1/energyConsumption*
  - Energy consumption for all houses in Hursley
    - *UK/Hants/Hursley/**+**/**+**/energyConsumption*

  - Multi-level wildcards **"#"** must appear at the end of the string

  For example:
  - Details of energy consumption, solar and alarm for all houses in SO212JN
    - *UK/Hants/Hursley/SO212JN/**#***

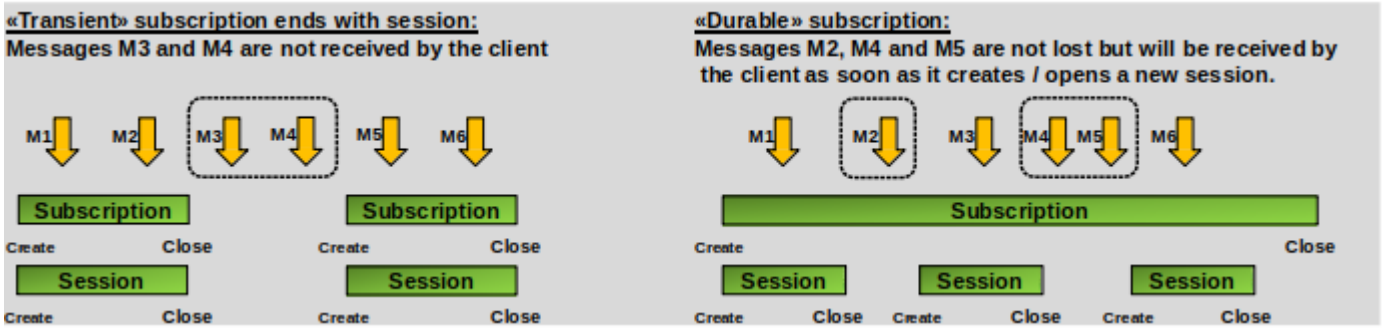
NOTE :

- Wildcards must be **next to a separator**
- **Cannot** be used wildcards when **publishing**

Publish Subscribe Messaging

- A subscription can be durable or non durable

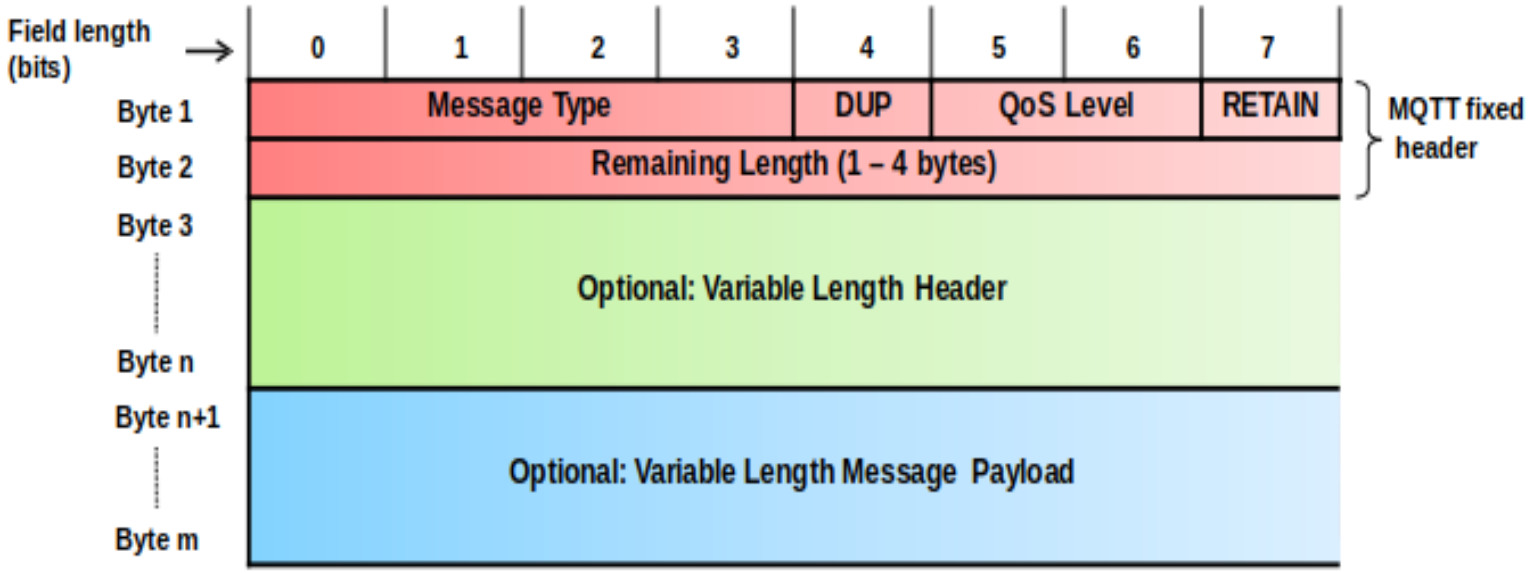
- Durable:
  - Once a subscription is in place a broker will forward matching messages to the subscriber:
    - Immediately if the subscriber is connected.
    - If the subscriber is not connected messages are stored on the server/broker until the next time the subscriber connects.



- Non-durable(Transient):
  - The subscription lifetime is the same as the time the subscriber is connected to the server / broker

MQTT message format

- MQTT messages contain a mandatory **fixed-length header (2 bytes)** and an optional message- specific variable length header and message payload.
- MQTT uses network byte and bit ordering.



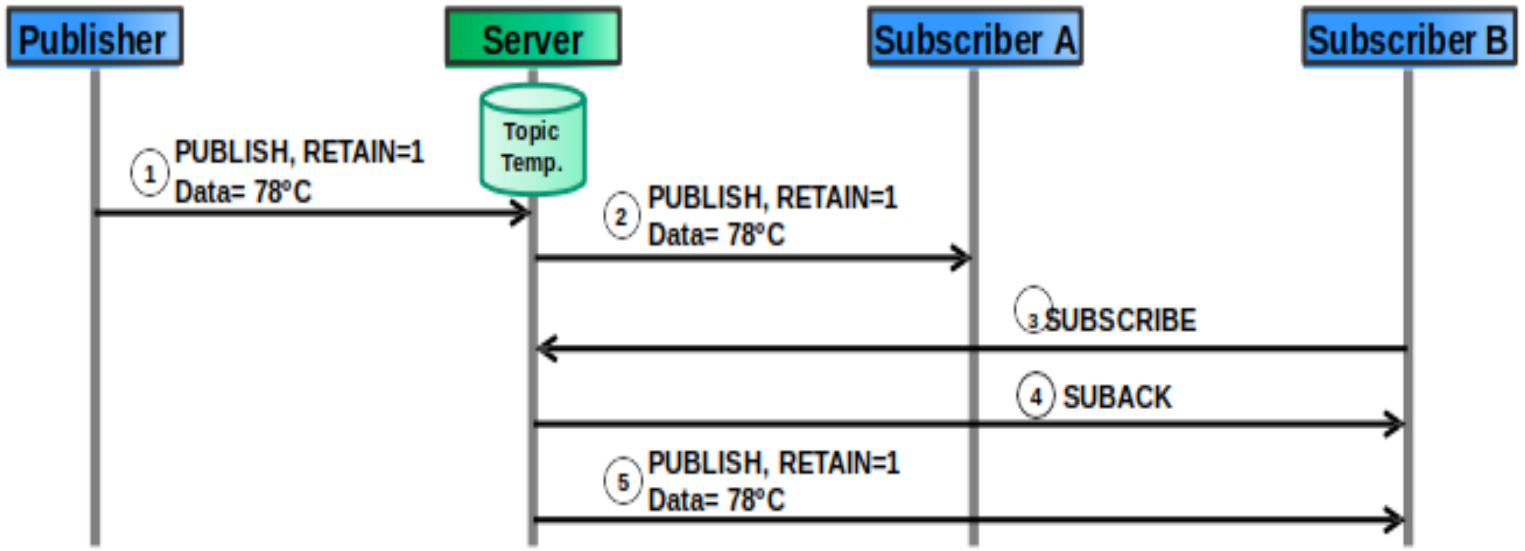
Overview of fixed header fields

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.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions.	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload.	

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.



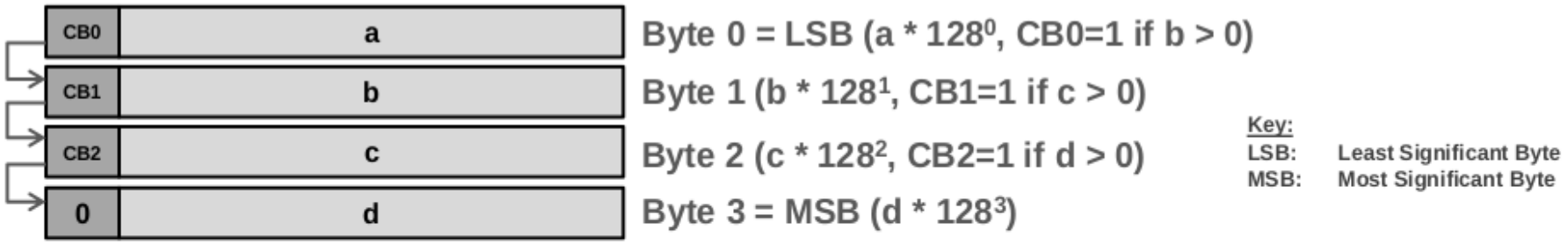
Remaining length (RL)

The remaining length field encodes the sum of the lengths of:

1. **(Optional)** variable length header
2. **(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 $\mathbf{a * 128^0 + b * 128^1 + c * 128^2 + d * 128^3}$ and placed into the RL field bytes as follows:

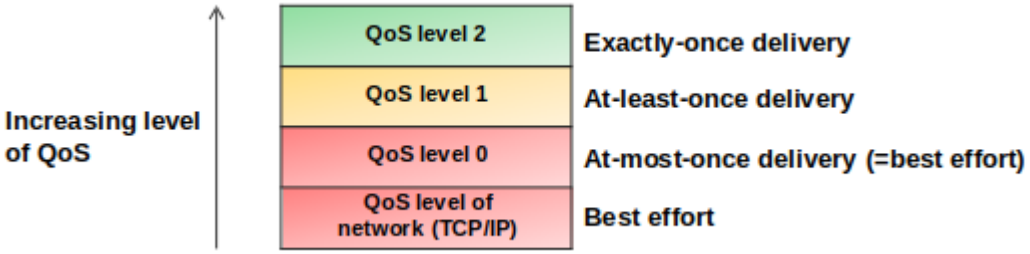


Example 1: **RL = 364 = 108*128⁰+2*128¹ → a=108, CB0=1, b=2, CB1=0, c=0, d=0, CB2=0**

Example 2: **RL = 25'897 = 41*128⁰ + 74*128¹ + 1*128² → a=41, CB0=1, b=74, CB1=1, c=1, CB2=0, d=0**

MQTT QoS

- 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.

MQTT QoS

QoS level 1:

- **At-least-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 **loss-lessly** to subscribers (e.g. alarming function). Applications simply discard duplicate messages by evaluating the message ID field.

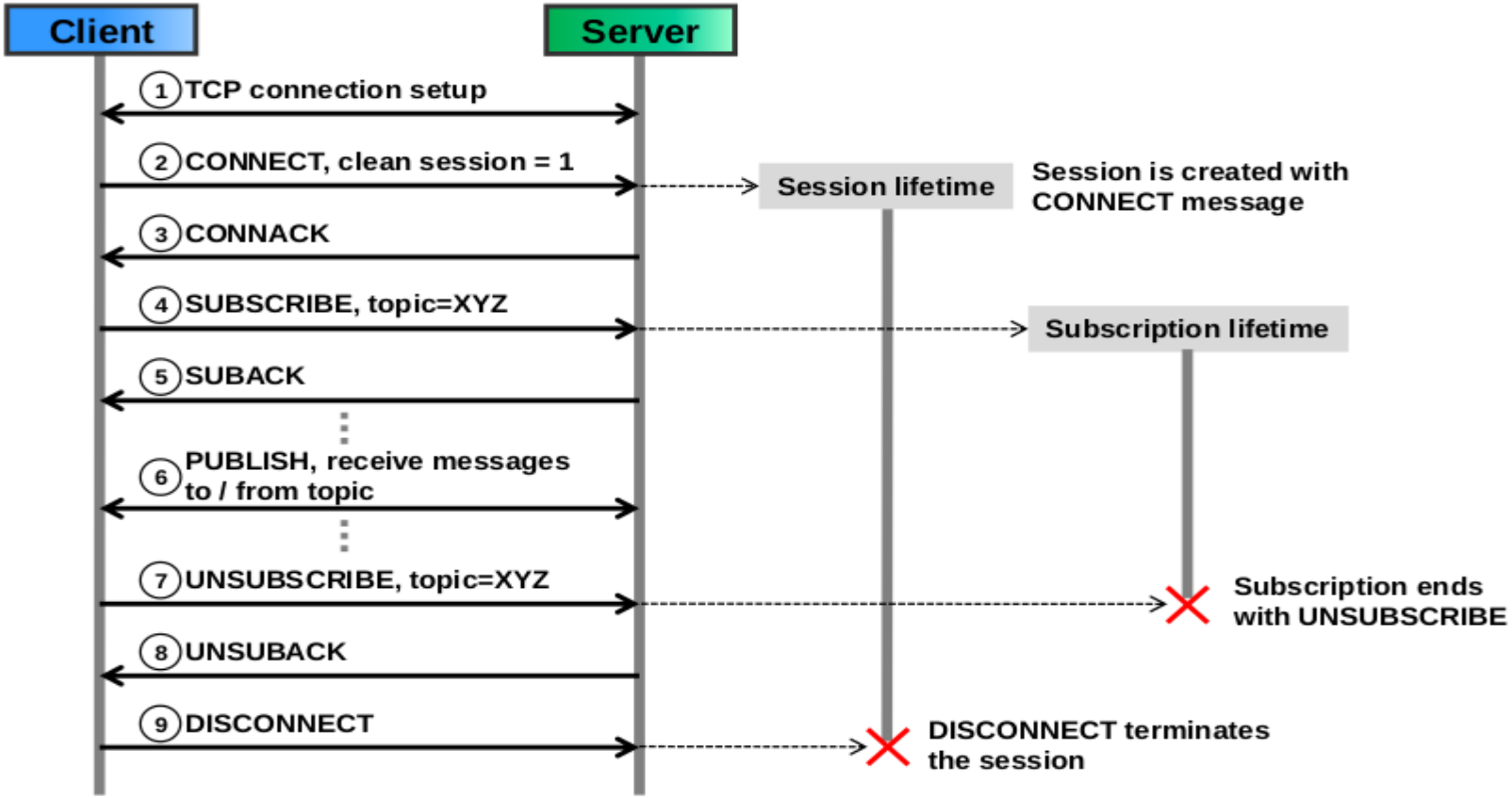
QoS level 2:

- **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.

CONNECT and SUBSCRIBE msg flow

Case 1:

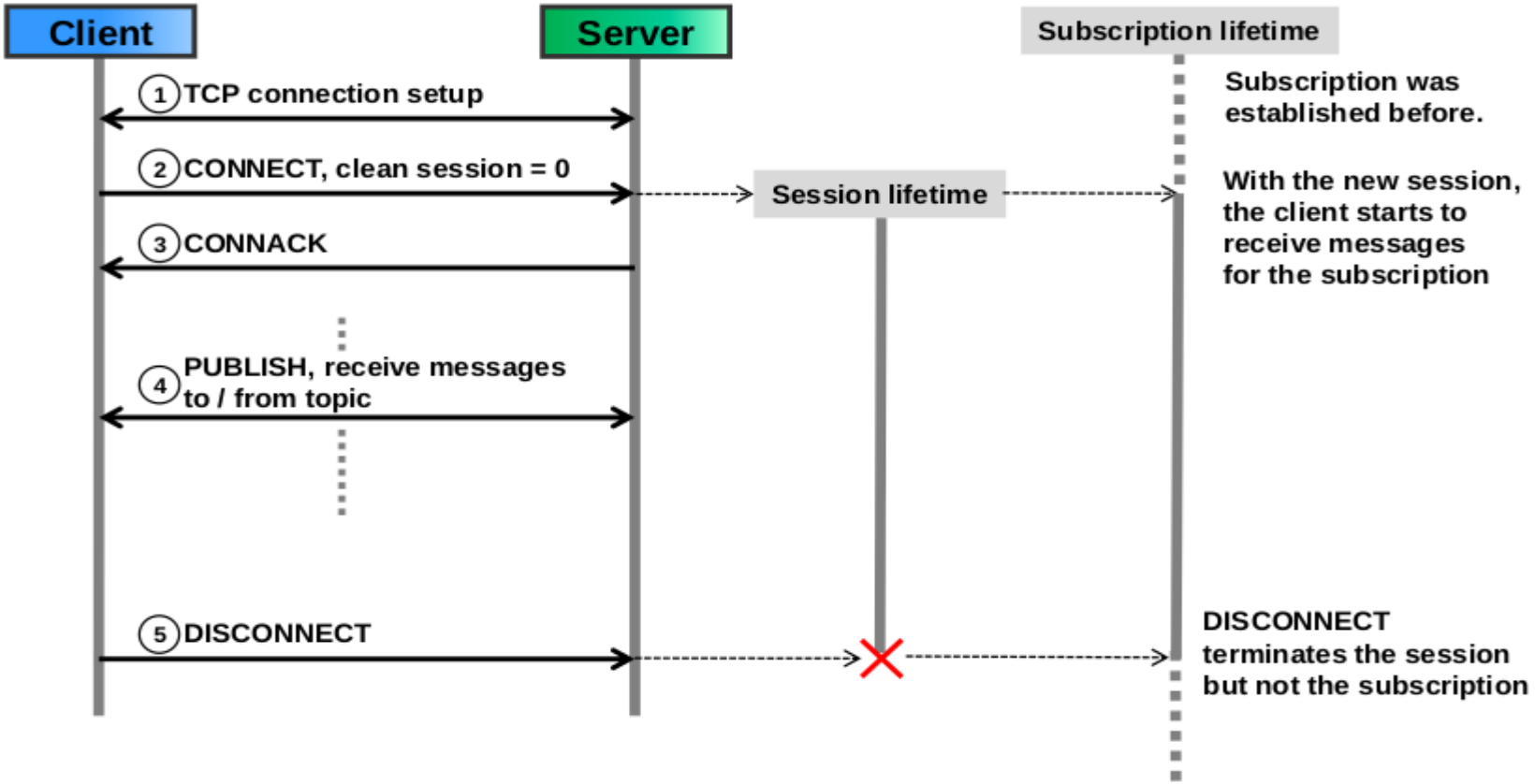
- Session and subscription setup with clean session flag = 1 («transient» subscription)



Cont…

Case 2:

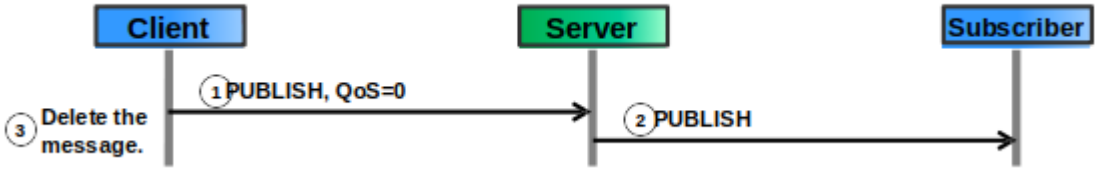
- Session and subscription setup with clean session flag = 0 («durable» subscription)



PUBLISH msg flows

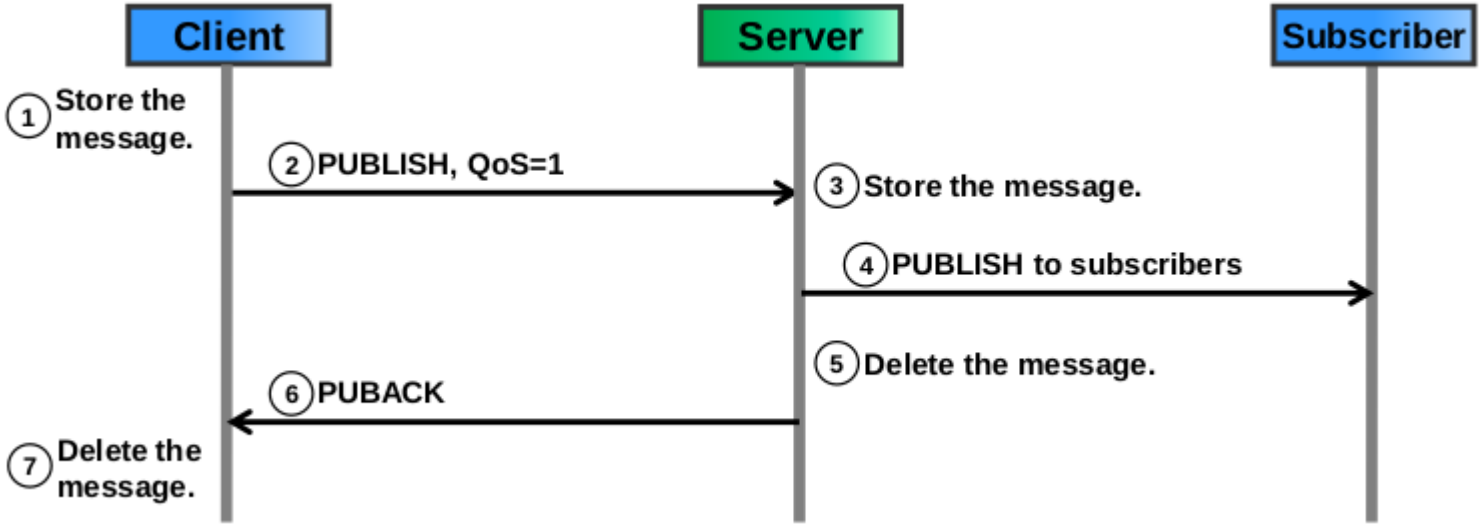
QoS level 0:

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



QoS level 1:

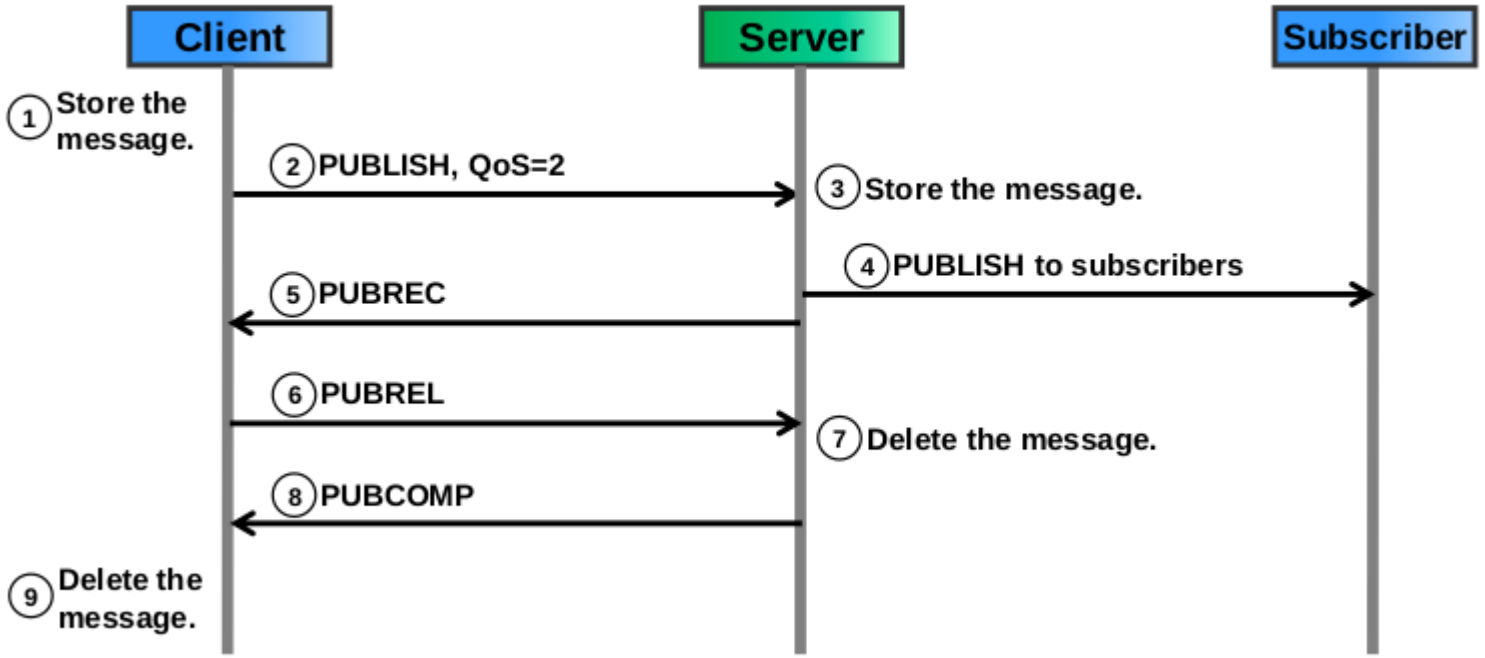
- QoS level 1 affords **at-least-once delivery semantics.** If the client does not receive the **PUBACK** in time, it **re-sends** the message.



Cont…

QoS level 2:

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



Thanks!