MQTT

Core Concepts & Developments Setup

Core Concepts

Broker: The central server that routes messages between clients (like a post office).

Client: Any device (sensor, app, microcontroller) that connects to the broker.

Topic: A UTF-8 string used to organize messages. Think of it like a "channel".

Publish: Sending a message to a topic.

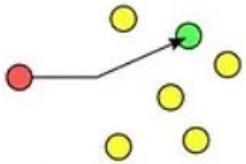
Subscribe: Registering to receive messages on a topic.

TCP

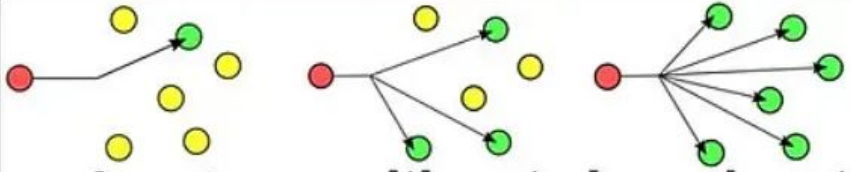
- **Slower but reliable transfers**
- **Typical applications:**
  - Email
  - Web browsing

UDP

- **Fast but non-guaranteed transfers ("best effort")**
- **Typical applications:**
  - VoIP
  - Music streaming

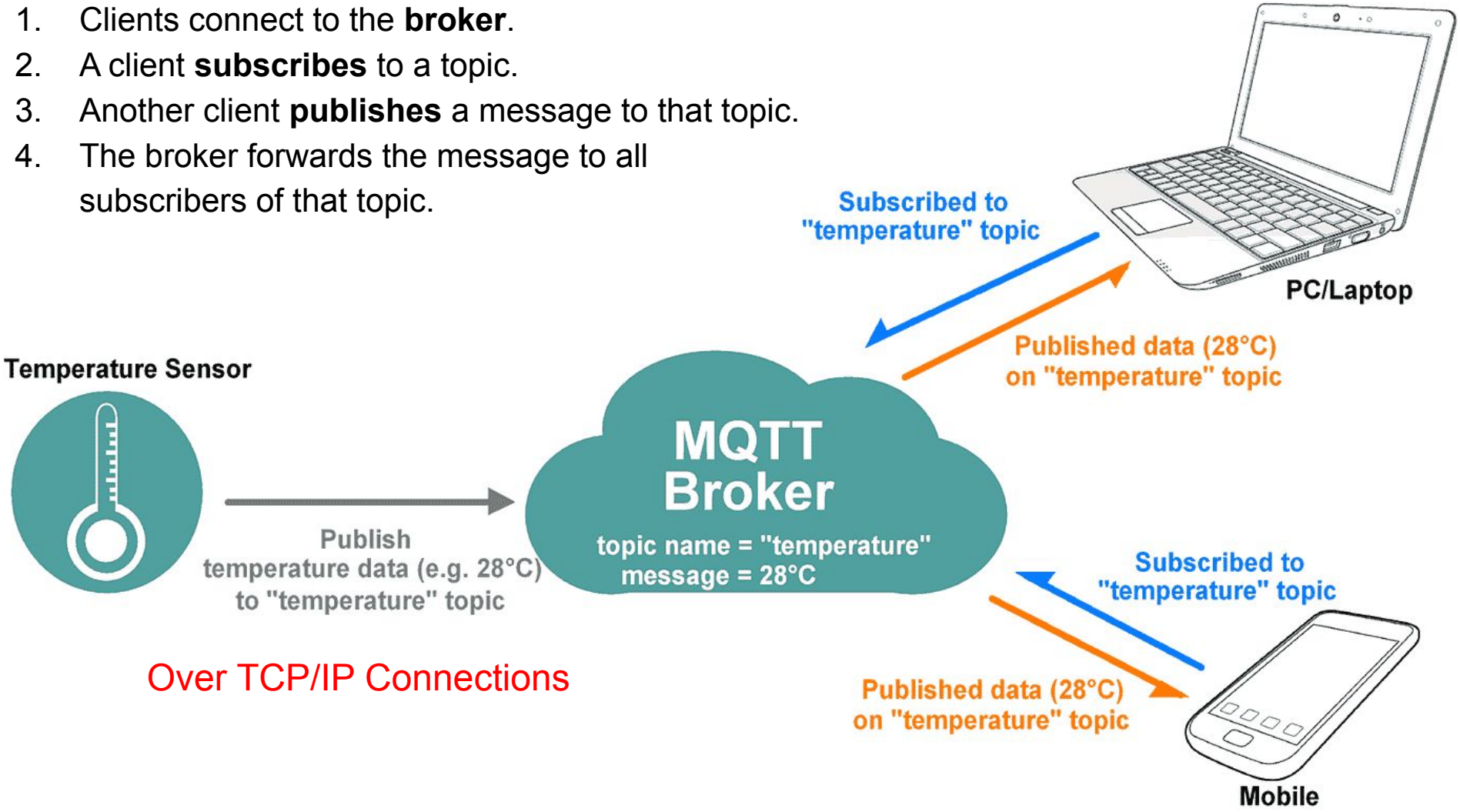


unicast



unicast **multicast** **broadcast**

1. Clients connect to the **broker**.
2. A client **subscribes** to a topic.
3. Another client **publishes** a message to that topic.
4. The broker forwards the message to all subscribers of that topic.



Over TCP/IP Connections

Security Levels of MQTT

Username/Password Authentication

TLS/SSL Encryption for secure communication

Access Control (topic-level restrictions)

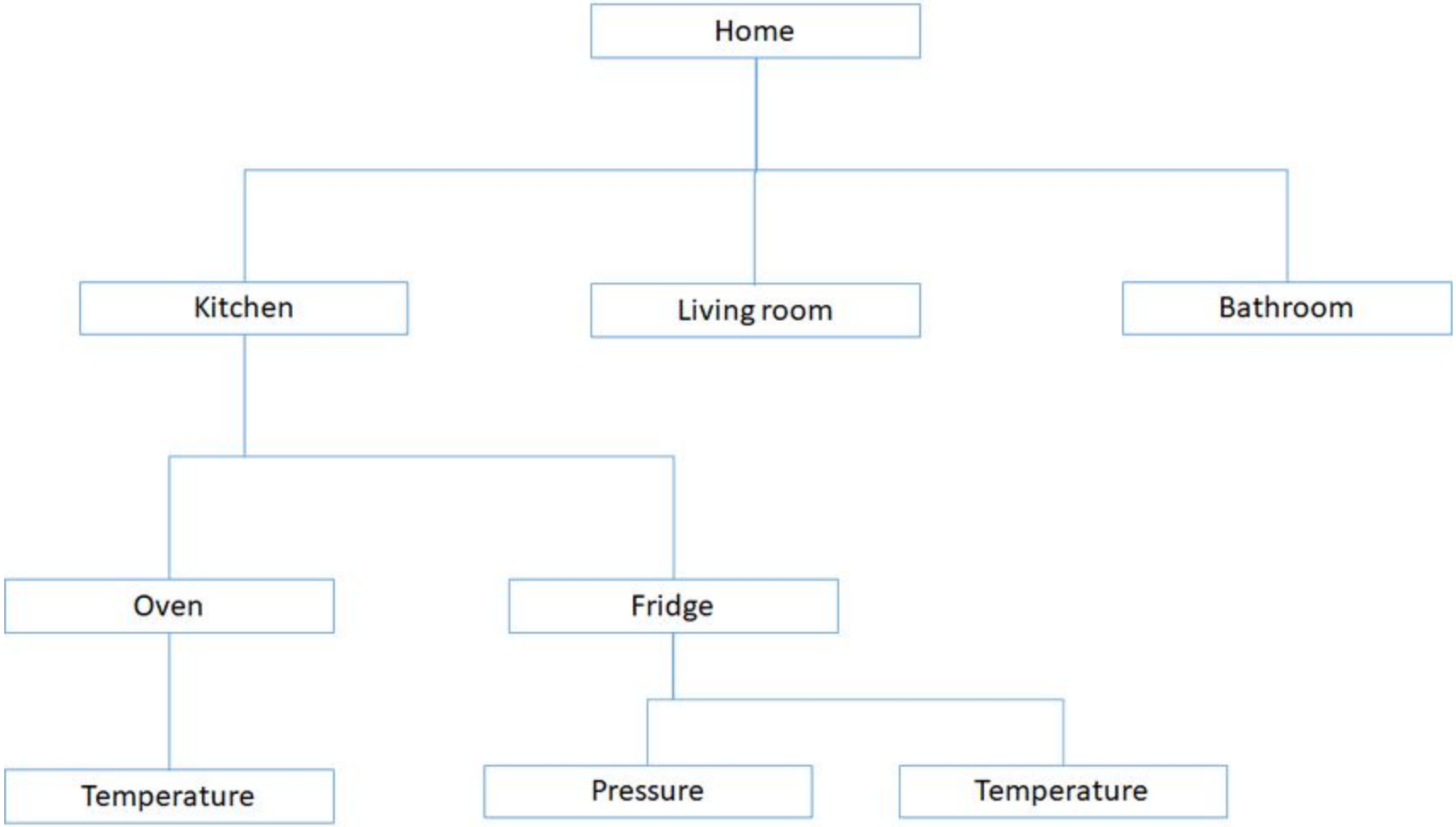
🧩 MQTT Topic Hierarchies and Wildcards

Topics are organized in a hierarchical format using `/`:

- `home/livingroom/temperature`

Wildcards:

- `+` (single-level): `home/+/temperature` matches `home/kitchen/temperature`
- `#` (multi-level): `home/#` matches `home/anything/here`


Home
Kitchen
Living room
Bathroom
Oven
Fridge
Temperature
Pressure
Temperature

1. Retained Messages

- Broker keeps the **last retained message** on a topic.
- New subscribers immediately receive it upon subscribing.

2. Last Will and Testament (LWT)

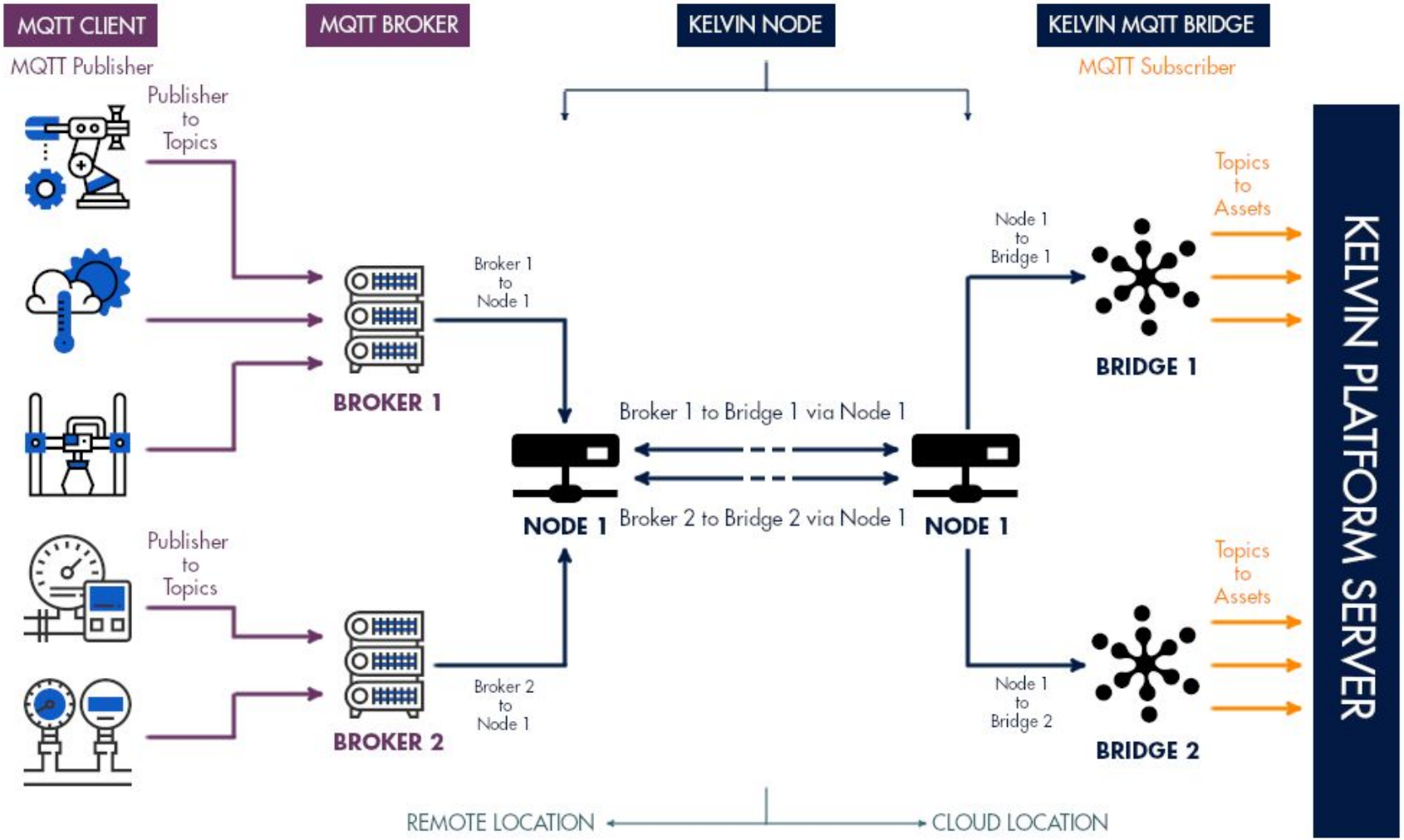
- A message defined at connection time to be sent by the broker **if the client disconnects unexpectedly**.

3. Persistent Sessions

- Allows clients to resume subscriptions and message queues **after reconnecting**.

4. Bridging Brokers

- You can connect multiple MQTT brokers together to share messages across networks or geographic locations.


MQTT CLIENT
MQTT BROKER
KELVIN NODE
KELVIN MQTT BRIDGE
MQTT Publisher
MQTT Subscriber
Publisher to Topics
Publisher to Topics
Broker 1 to Node 1
Broker 2 to Node 1
BROKER 1
BROKER 2
NODE 1
Broker 1 to Bridge 1 via Node 1
Broker 2 to Bridge 2 via Node 1
NODE 1
Node 1 to Bridge 1
Node 1 to Bridge 2
BRIDGE 1
BRIDGE 2
Topics to Assets
Topics to Assets
KELVIN PLATFORM SERVER
REMOTE LOCATION
CLOUD LOCATION

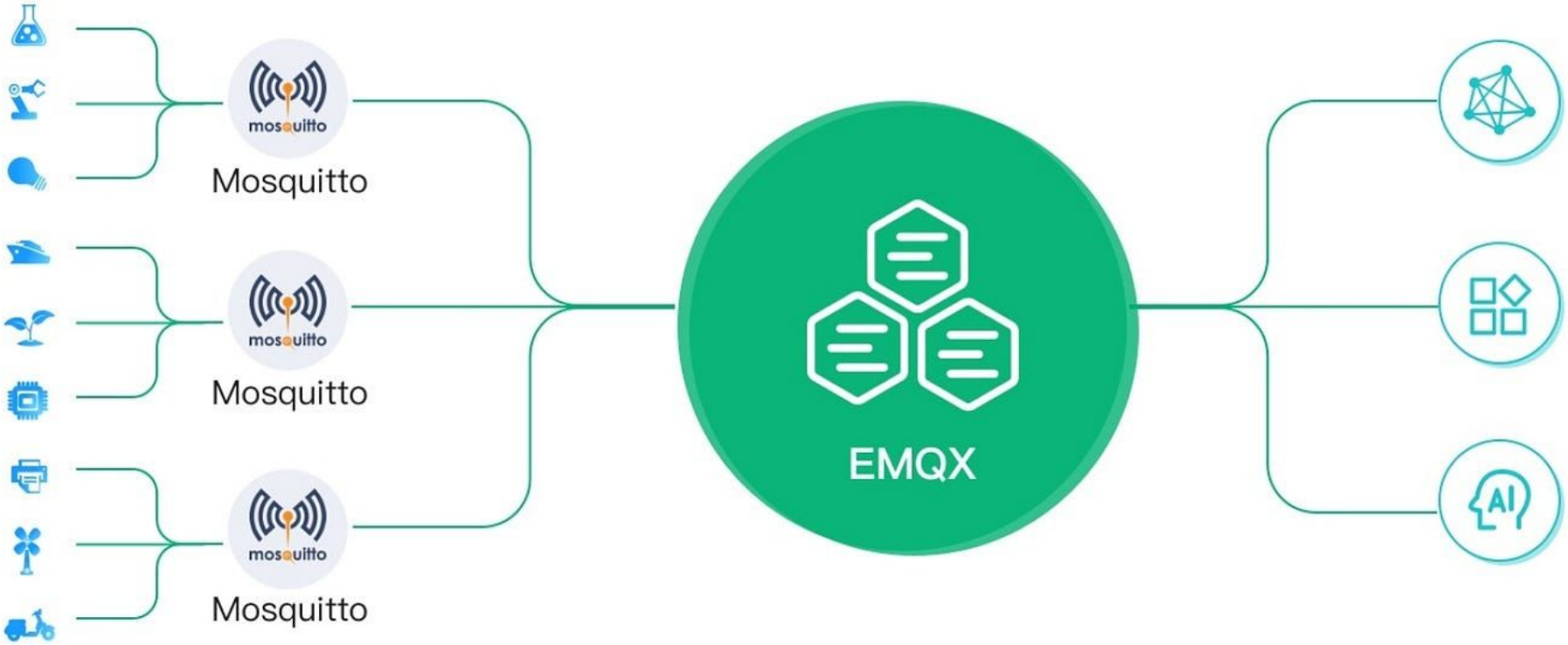
Tools

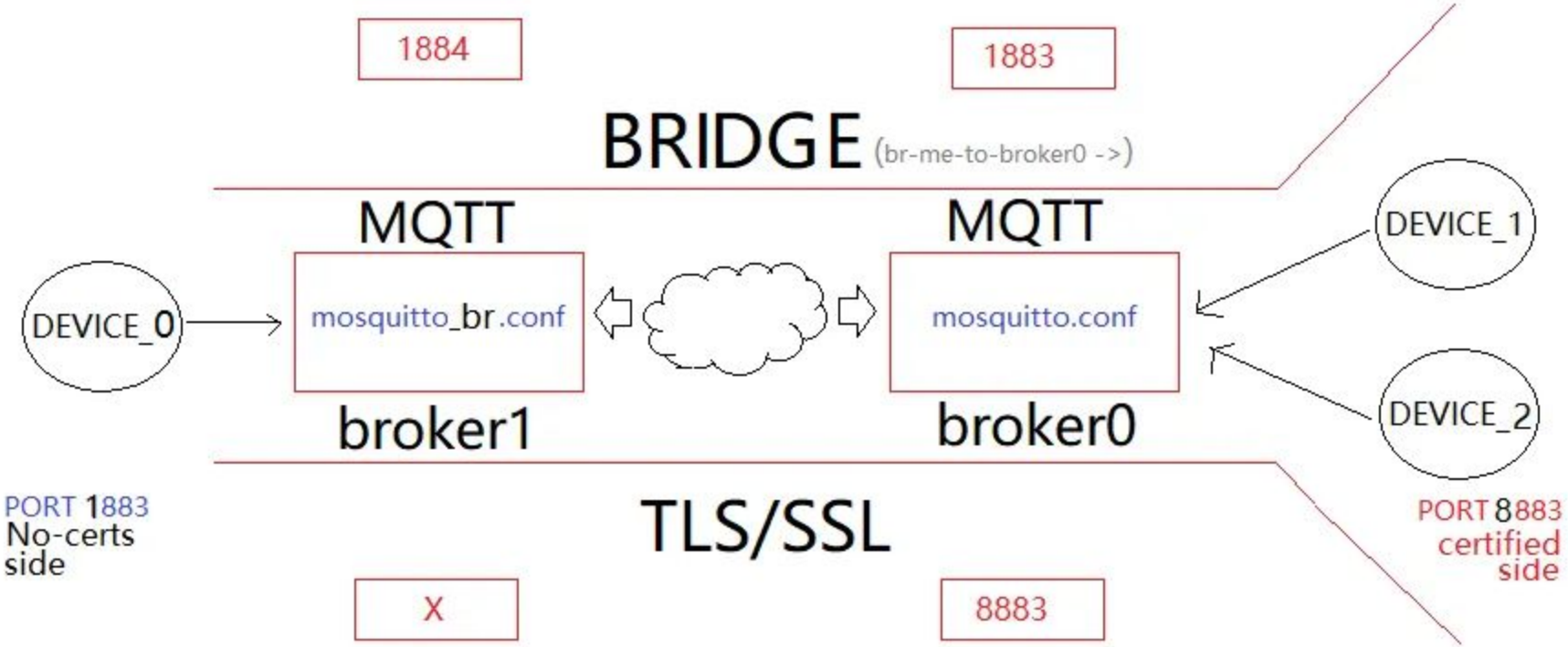
Brokers: Mosquitto, EMQX, HiveMQ, VerneMQ

Clients: MQTT.fx, MQTT Explorer, mosquitto_pub/sub, Node-RED, custom code (Python `paho-mqtt`, C++, etc.)

Command Line Tools:

```
bot@bot-x:~$ mosquitto_ctrl   mosquitto_pub   mosquitto_sub   mosquitto_passwd  mosquitto_rr
```


mosquitto
Mosquitto
mosquitto
Mosquitto
mosquitto
Mosquitto
EMQX
AI


1884
1883
BRIDGE (br-me-to-broker0 ->)
MQTT
MQTT
DEVICE_0
mosquitto_br.conf
mosquitto.conf
DEVICE_1
DEVICE_2
broker1
broker0
PORT 1883
No-certs
side
TLS/SSL
PORT 8883
certified
side
X
8883

Installation Broker on Linux System

- # sudo apt install **mosquitto mosquitto-clients**
- # sudo systemctl status mosquitto
- # sudo systemctl enable mosquitto
- # sudo systemctl start mosquitto

- Configure Broker ??

Publish / Subscribe Testing (With mqtt_cli Client)

kyawswarhtun — bot@bot-x: /etc/mosquitto/conf.d — ssh bot@192.168.2.10 — 145×

~ — bot@bot-x: /etc/mosquitto/conf.d — ssh bot@192.168.2.10

```
bot@bot-x:/etc/mosquitto/conf.d$ mosquitto_sub -h localhost -t test
hello world
hello world
hello world2
hello world3
hello world4
```

kyawswarhtun — bot@bot-x: ~ — ssh bot@192.168.2.10 — 80×24

~ — bot@bot-x: ~ — ssh bot@192.168.2.10

```
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world2"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world3"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world4"
bot@bot-x:~$
```

Configuration on Local & Remote Host

```
bot@bot-x:/etc/mosquitto$ ls
aclfile.example  ca_certificates  certs  conf.d  mosquitto.conf  pskfile.example  pwfile.example
```

Set Auth (username, password)

```
bot@bot-x:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd bot
Password:
Reenter password:

bot@bot-x:~$ cat /etc/mosquitto/passwd
bot:$7$101$aQP+Yw1np7NbyAIb$fgDivKODODB/4Ikn7YyjP/v+R1276uyx4K4FLe/BVIa4IHOAUeFvQyBvxoO
sx3zH7W80jImBXIz3IJa0KPUWvQ==
```

```
  GNU nano 7.2                  /etc/mosquitto/mosquitto.conf *
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

### start userconfig ###

allow_anonymous false
password_file /etc/mosquitto/passwd


^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

Testing With New Configuration

Reload Configuration..

```
# sudo systemctl restart mosquitto
```

```
$ mosquitto_sub -h localhost -t test -u "bot" -P "mosquitto"

$ mosquitto_pub -h localhost -t test -m "hello world 1" -u "bot" -P "mosquitto"
```

```
^Cbot@bot-x:/etc/mosquitto/conf.dmosquitto_sub -h localhost -t test -u "bot" -P "mosquitto"
hello world 1
hello world 2
hello world 3
hello world 4
hello world 5
```

kyawswarhtun — bot@bot-x: ~ — ssh bot@192.168.2.10 — 80×24

~ — bot@bot-x: ~ — ssh bot@192.168.2.10

```
bot@bot-x:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd bot
Password:
Reenter password:
bot@bot-x:~$ less /etc/mosquitto/passwd
bot@bot-x:~$ cat /etc/mosquitto/passwd
bot:$7$101$aQP+Yw1np7NbyAIb$fgDivKODODB/4Ikn7YyjP/v+R1276uyx4K4FLe/BVIa4IHOAUeFv
QyBvxoOsx3zH7W80jImBXlz3lJa0KPUWvQ==
bot@bot-x:~$ sudo nano /etc/mosquitto/mosquitto.conf
bot@bot-x:~$ sudo systemctl restart mosquitto
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 1" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 2" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 3" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 4" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 5" -u "bot" -P "
mosquitto"
bot@bot-x:~$
```

QoS defines the reliability of message delivery:

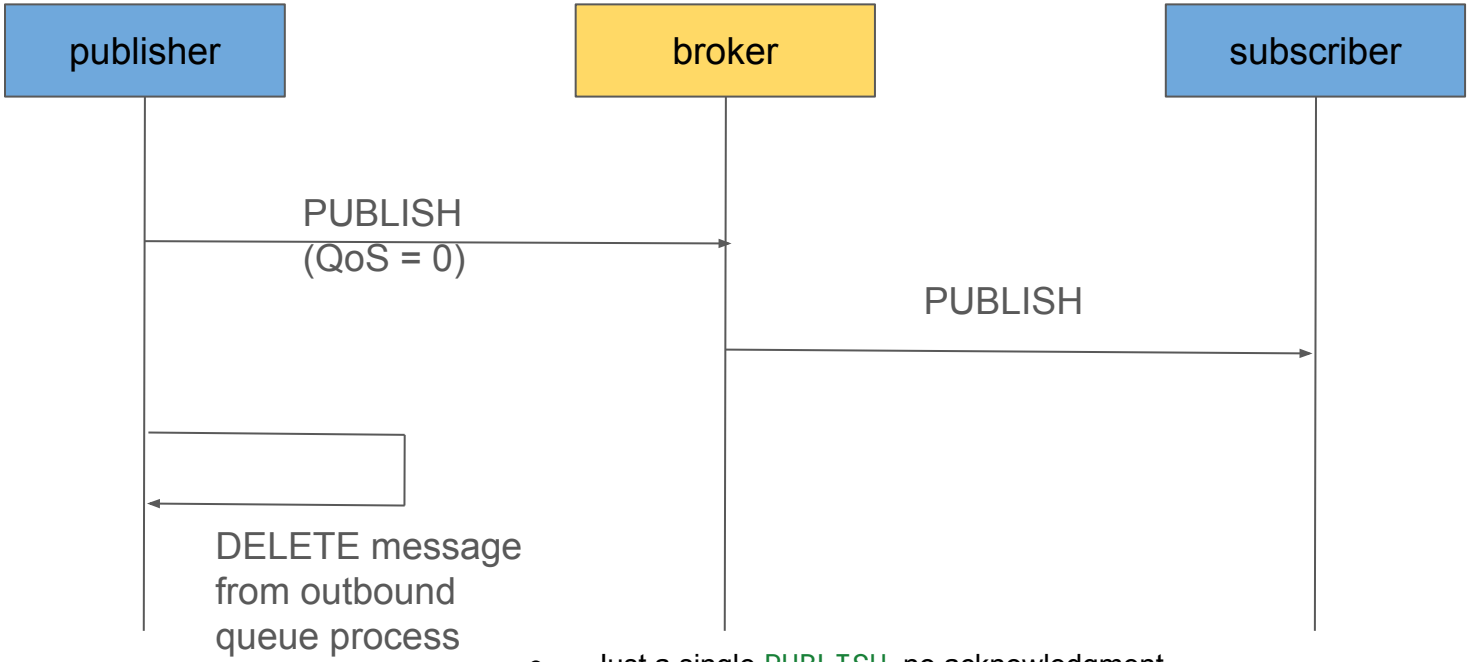
QoS 0: when we prefer that the message will arrive at most once; the message will be received or it won't, there isn't a chance of a duplicate; at most once; fire and forget; the most unreliable transfer mode.

QoS 1: when we want the message to arrive at least once but don’t care if it arrives twice (or more); at least once;

QoS 2: when we want the message to arrive exactly once. A higher QOS value means a slower transfer; exactly once.

MQTT : Quality Of Service

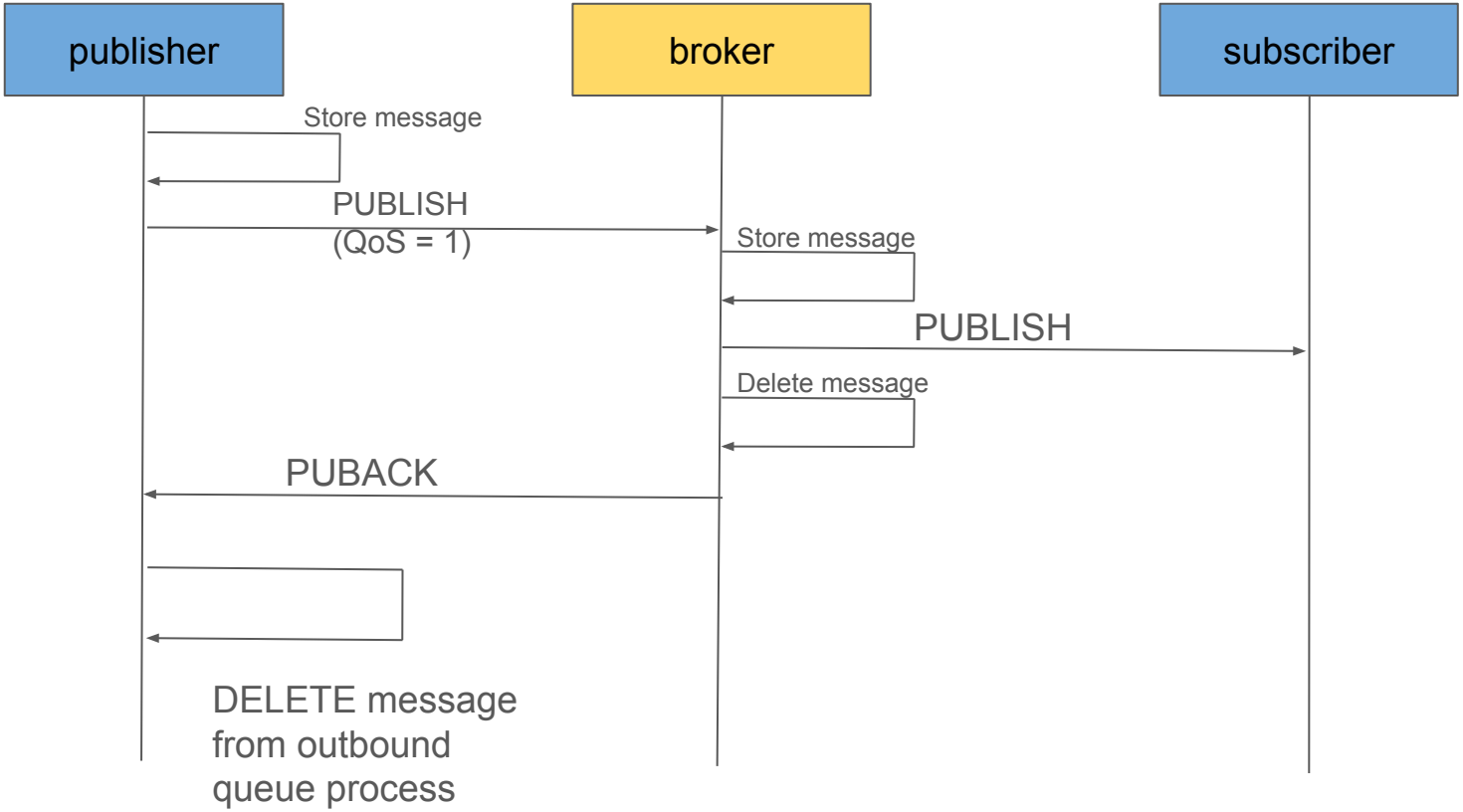
QoS 0 : At Most Once (fire and forget)

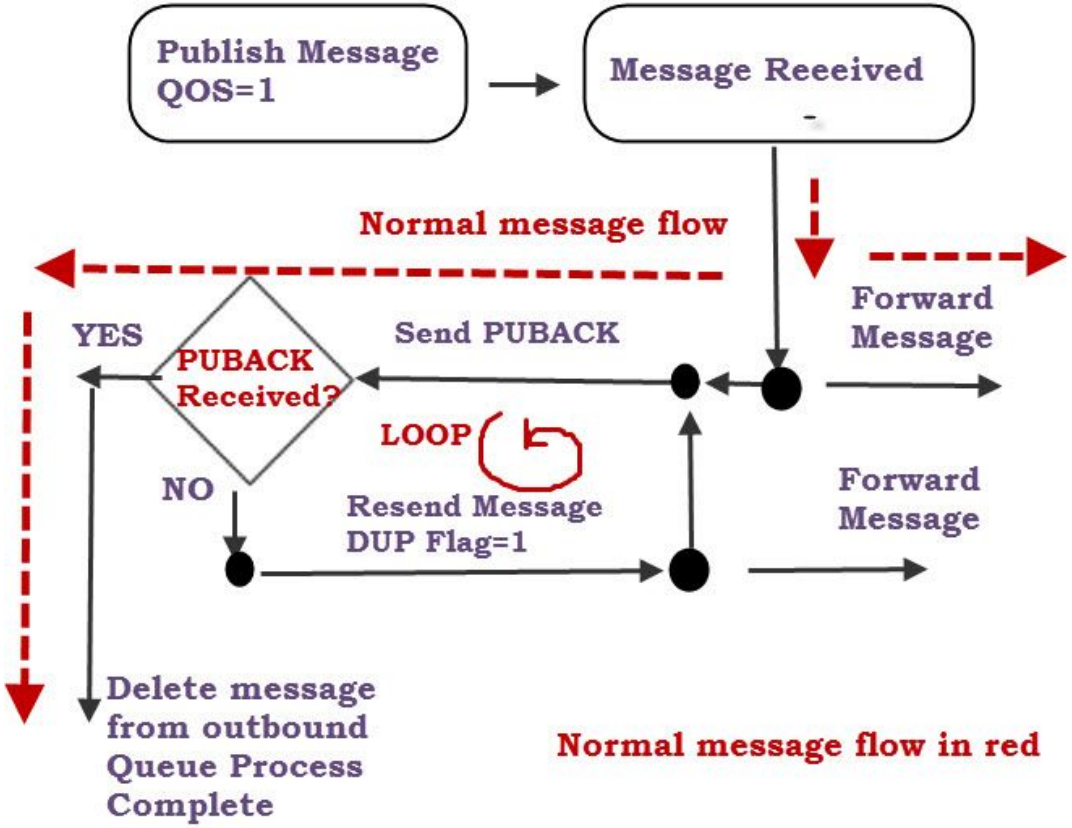


- Just a single `PUBLISH`, no acknowledgment.

MQTT : Quality Of Service

QoS 1 : At Least Once





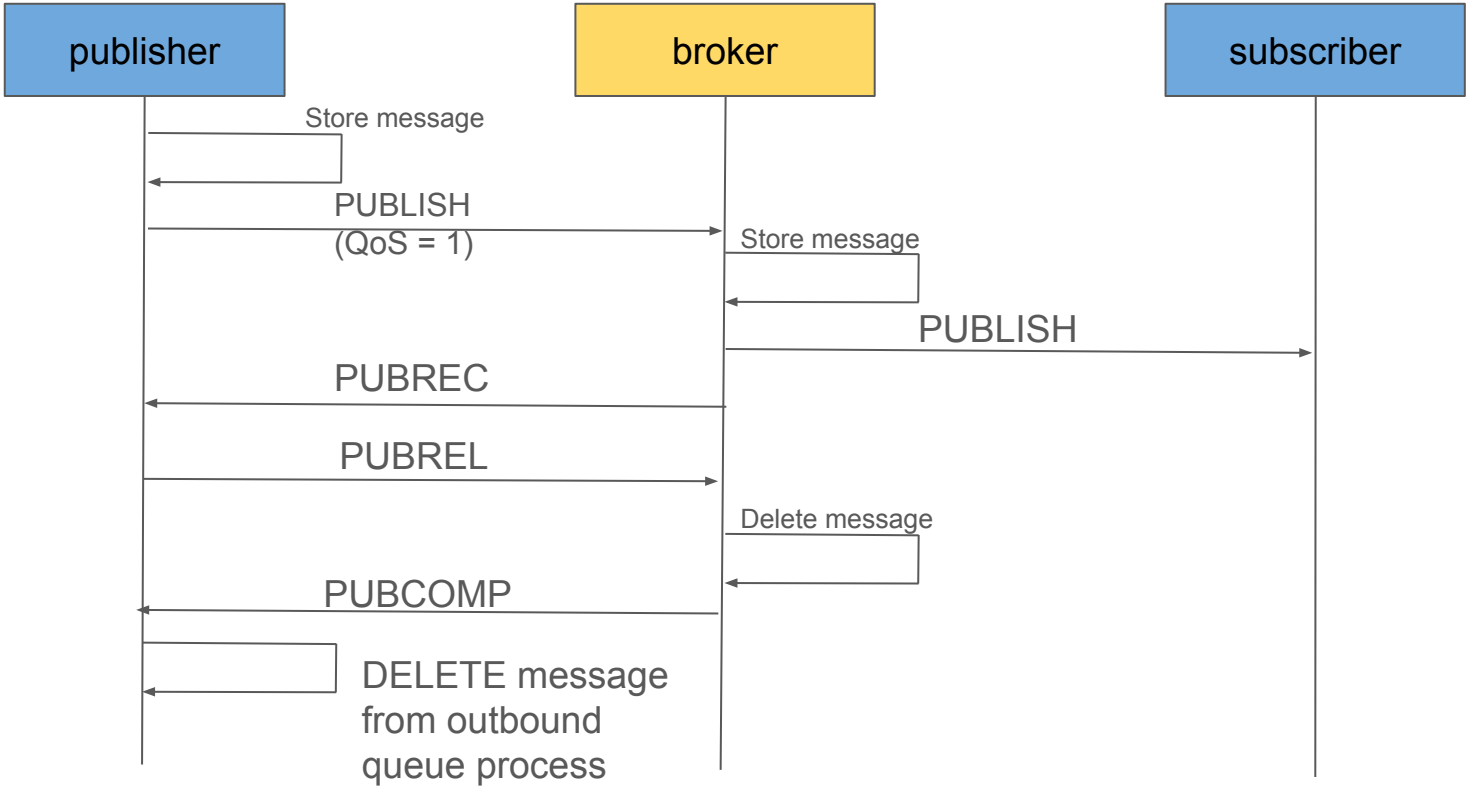
MQTT QOS 1 Message Flow Diagram

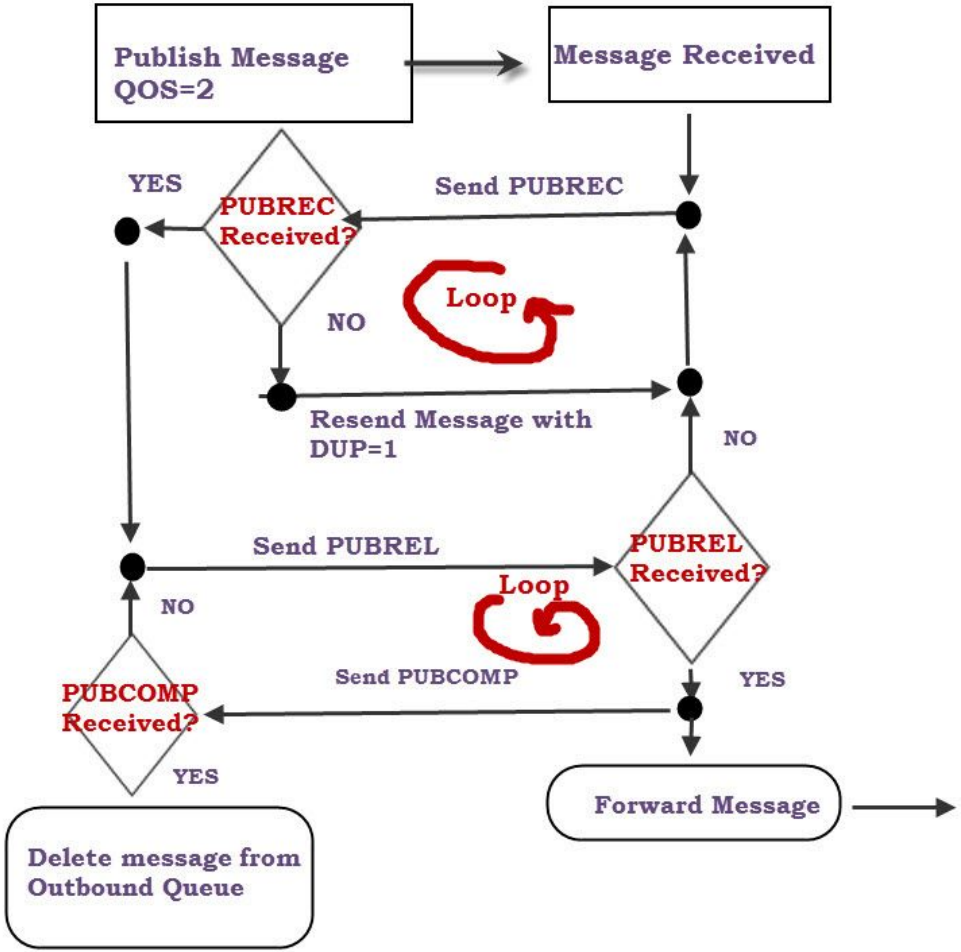
QoS 1:

- PUBLISH → wait for PUBACK.

MQTT : Quality Of Service

QoS 2 : Exactly Once





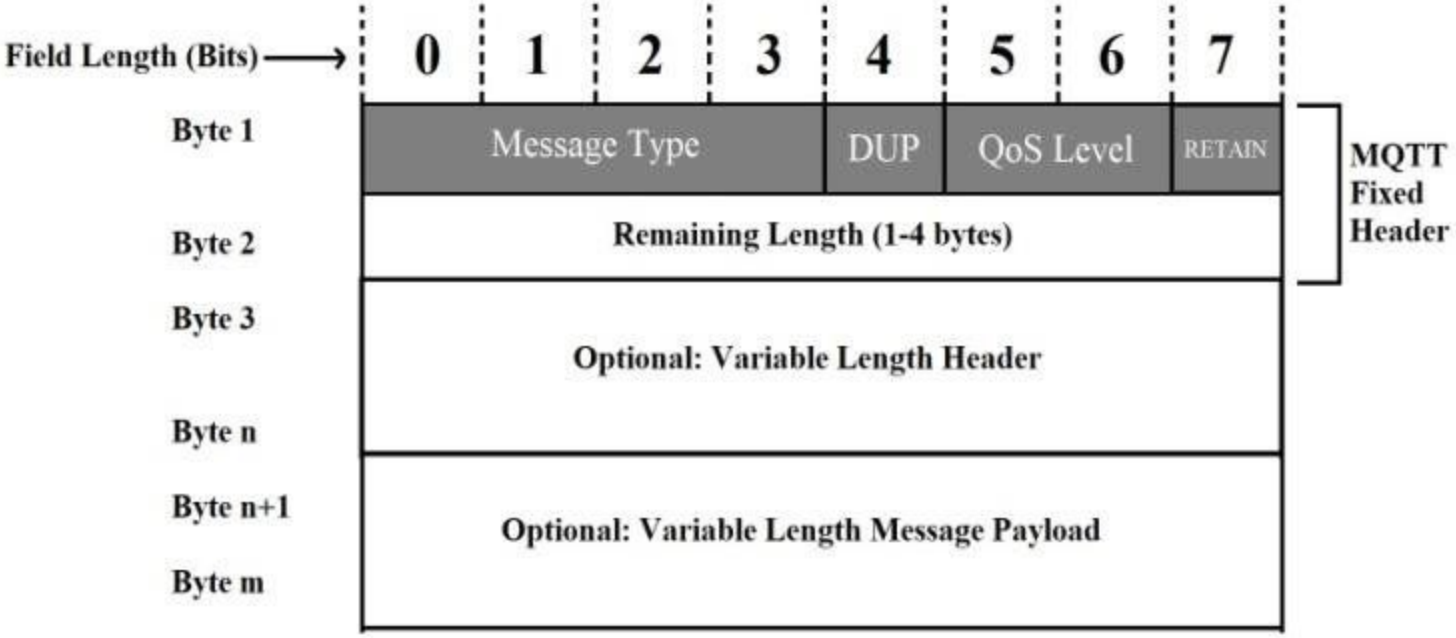
MQTT QOS 2 Message Flow Diagram

QoS 2:

- Four-way handshake:
  - `PUBLISH` → `PUBREC` → `PUBREL` → `PUBCOMP`
- Guarantees **no duplicate delivery**.

MQTT Packet Flow (Core Protocol Packets)

Packet	Purpose
`CONNECT`	Client to broker, starts a session
`CONNACK`	Broker to client, acknowledges connection
`PUBLISH`	Send a message to a topic
`PUBACK`	QoS 1 message acknowledgment
`PUBREC`	QoS 2 – Message received (part 1)
`PUBREL`	QoS 2 – Message released (part 2)
`PUBCOMP`	QoS 2 – Completion confirmation
`SUBSCRIBE`	Client subscribes to a topic
`SUBACK`	Subscription acknowledgment
`UNSUBSCRIBE`	Client unsubscribes
`UNSUBACK`	Acknowledge unsubscribe
`PINGREQ`	Client heartbeat check
`PINGRESP`	Broker heartbeat response
`DISCONNECT`	Graceful session close



Message Type: PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP
Qos: 0/1/2
DUP: DUPLICATE FLAG
RETAIN: Set to ON to store last known value

Persistent Configuration of Broker (bridge extensions)

MQTT is dynamic by design – topics are not “pre-declared” like in AMQP (e.g., RabbitMQ).

But you **can enforce topic-level control** and simulate structure.

Thank You

CLIENT NAME XX.XX.XX

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam *nonummy* nibh euismod tincidunt ut laoreet dolore magna aliquam erat *volutpat.*