MQTT

Version 5.0 · verified 2026-08-03

Pub/sub messaging for unreliable networks — sessions, topics, QoS, retained messages, wills, and MQTT 5 additions.

Contents

Mental model

MQTT is publish/subscribe through a broker: clients never address each other, they publish to a topic string and the broker forwards to every client whose filter matches. Nothing is declared in advance — a topic exists because someone published to it. The protocol's real subject is the **session**: broker-side state that outlives the TCP connection, and the only reason a device on a flaky link doesn't lose what it missed.

Connections and sessions

CONNECT field	Effect
Client Identifier	Names the session; reuse to resume
Clean Start	`1` discards any existing session
Session Expiry Interval	Seconds it outlives the socket
Keep Alive	Seconds; broker cuts you at 1.5×
Will fields	Published if the link drops

A client opens TCP on 1883 (8883 for TLS), sends CONNECT, and gets CONNACK with a reason code. Clean Start `0` **plus** a non-zero Session Expiry Interval is what makes subscriptions and queued QoS 1/2 messages survive a reconnect. Keep Alive is a promise to send something every N seconds; PINGREQ fills the silence.

```
# resume the session named sensor-42
mosquitto_sub -h broker.example -p 8883 \
```

```
  --cafile ca.crt -i sensor-42 -c \
  -t 'home/+/temp' -q 1
```

> **Gotcha:** Clean Start `0` alone is not persistence. An absent Session Expiry Interval defaults to `0`, and the session still dies with the connection, subscriptions included.

Topics and wildcards

Filter	Matches	Does not match
`home/+/temp`	`home/kitchen/temp`	`home/a/b/temp`
`home/#`	`home/a/b/c`	`house/a`
`+/temp`	`attic/temp`	`temp`

Topics are UTF-8 strings split on `/`. Publishers use a topic *name*, never containing a wildcard; subscribers use a topic *filter*, which may. `+` stands for exactly one level, `#` for all remaining levels and must be last. Nothing validates a topic, so a typo silently creates one that nobody reads.

```
home/kitchen/temp    publish: a topic NAME
home/+/temp          subscribe: one level
home/#               subscribe: rest of the tree
$SYS/broker/uptime   the broker's own stats
```

> **Gotcha:** Wildcards never reach `$` topics. The server must not match a filter starting with `#` or `+` against a name starting with `$`, so subscribing to `#` does **not** subscribe you to `$SYS`.

Packets

Packet	Purpose
CONNECT / CONNACK	Open a session, report the outcome
PUBLISH	Carry one application message
SUBSCRIBE / SUBACK	Add filters; a code per filter
PINGREQ / PINGRESP	Prove the link is alive
DISCONNECT	Close cleanly, with a reason code

Fifteen control packet types exist; a session is a conversation in them. PUBLISH is one-way — beyond the QoS acknowledgements a publisher learns nothing, not even whether anyone was subscribed. SUBACK returns a reason code *per filter*, so a partly rejected SUBSCRIBE is normal.

```
mosquitto_pub -h broker.example \
  -t 'home/kitchen/temp' -m '21.5' -q 1
```

> **Gotcha:** Nothing queues for a client that is not already subscribed. Offline delivery works only because the session still holds the subscription — a first-time connection gets nothing published before it arrived.

Quality of service

QoS	Guarantee	Cost
`0`	At most once	One packet, no state
`1`	At least once	PUBACK; duplicates possible
`2`	Exactly once	Four packets, two round trips

QoS is per hop, not end to end. The publisher's QoS covers publisher→broker; delivery to a subscriber is the minimum of the published QoS and the maximum granted to that subscription. Pick per message: `0` for a reading sent every second, `1` for a command, `2` only where a duplicate does damage.

```
# subscriber caps delivery at QoS 1
mosquitto_sub -t 'home/#' -q 1
```

> **Gotcha:** QoS 1 means *at least* once — duplicates are correct protocol behavior, not a broker bug. Make handlers idempotent or deduplicate on an application-level id.

Retained messages

RETAIN flag	Broker does
`1`, with payload	Stores it as the topic's last known value
`1`, zero bytes	Deletes the stored message
`0`	Delivers to current subscribers only

A retained message answers “what is the current value?” — the broker keeps exactly one per topic and hands it to each new subscriber immediately, so a dashboard renders at once instead of waiting for the next reading. It is a last-value cache, not a queue: the next retained publish replaces the previous one.

```
# set the last known value
mosquitto_pub -t 'home/kitchen/temp' -r -m '21.5'

# clear it — zero-byte retained publish
mosquitto_pub -t 'home/kitchen/temp' -r -n
```

> **Gotcha:** Retained state belongs to the topic, not the publisher. A decommissioned device leaves its last reading live on the broker forever unless something clears it.

Last Will and Testament

Will field	Purpose
Will Topic / Payload	What to announce on an unclean exit
Will QoS / Retain	How it is delivered
Will Delay Interval	Grace period before publishing

The client hands the broker a message at CONNECT; the broker publishes it if the connection ends without a clean DISCONNECT. Pair it with a retained `online` message at startup and you have presence for

free. Will Delay Interval lets a reconnect cancel it — the fix for a flapping link that would otherwise announce its own death hourly.

```
mosquitto_sub -h broker.example -i sensor-42 \
  --will-topic 'home/sensor-42/status' \
  --will-payload 'offline' --will-retain \
  -c -t 'home/#'
```

> **Gotcha:** A DISCONNECT packet makes the broker discard the will without publishing it. That is the intent — but it means the will fires only on a crash or a dead link, and not until Keep Alive × 1.5 has elapsed.

MQTT 5 additions

Feature	Use it for
Reason codes	Why a packet was refused
User properties	Key/value headers on a message
Response Topic + Correlation Data	Request/response
Message Expiry Interval	Dropping stale queued messages
Shared subscriptions	Load-balancing one filter
Topic alias	Replacing a long topic with an integer

5.0 keeps the 3.1.1 wire model and adds the metadata it lacked: properties ride on the packet, so a message declares its own content type

and expiry, and acknowledgements carry a reason code instead of failing silently. Shared subscriptions are the architectural one — a group of clients becomes a consumer pool, each matching message going to exactly one member.

```
$share/<group>/<filter>
$share/workers/home/+/temp
```

> **Gotcha:** A shared subscription trades ordering for throughput. Per-topic order is guaranteed to a single subscriber, but spread across a group each member sees only its own slice, in its own order.

Further reading

- MQTT Version 5.0 — OASIS Standard
- MQTT Version 3.1.1 — OASIS Standard
- mqtt.org — FAQ
- Eclipse Mosquitto — man pages