Meshtastic MQTT Overview

Meshtastic's **MQTT module** extends a LoRa mesh network over the internet by bridging packets between the radio air interface and an MQTT message broker. A gateway node - a Meshtastic device with a working Wi-Fi or Ethernet connection - *uplinks* packets it hears over LoRa to the broker and optionally *downlinks* packets received from the broker back to the local mesh. This page explains the architecture, the public broker, and the trade-offs of the approach.

What MQTT Does in Meshtastic

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe protocol designed for constrained devices and unreliable networks. In the Meshtastic context it serves as the internet backbone that connects isolated mesh islands:

- **Uplinking** - the gateway node listens for LoRa packets from nearby nodes, re-serialises them as MQTT payloads, and publishes them to the broker under a well-known topic hierarchy. The canonical topic is `msh/REGION/2/e/CHANNELNAME/USERID` for raw/encrypted protobuf, or `msh/REGION/2/json/CHANNELNAME/USERID` for JSON. The `2/e` (or `2/json`) segment is the protocol version and encoding; it is not optional.
- **Downlinking** - the gateway subscribes to the same topic hierarchy; when the broker delivers a message published by a gateway in another location, the local gateway broadcasts it over LoRa to its nearby mesh.

The net effect: two mesh islands in different cities share a common channel as if they were on the same local RF network - provided both have an internet-connected gateway and use the same channel encryption key.

Public MQTT Server: mqtt.meshtastic.org

The Meshtastic project operates a community MQTT broker at `mqtt.meshtastic.org` on port `1883` (unencrypted) and port `8883` (TLS). This server is free to use for experimentation and for joining the global mesh.

- **Requires credentials** - the public broker is *not* anonymous. Connect with the default username `meshdev` and password `large4cats` (pre-filled by the firmware/CLI). These are

shared community credentials, not per-user accounts; external clients (mosquitto_sub, MQTT Explorer, Node-RED, Home Assistant) must supply them explicitly or the connection is rejected.
- **No privacy guarantee** - any message published to the public broker is visible to anyone who subscribes to the same topic.
- **Unencrypted at the MQTT layer** - if you use the default channel key (`AQ==`, which is the single byte `0x01` - not all-zeros), your messages are readable by any MQTT subscriber. Always configure a custom channel key for any sensitive or private use.

> **Privacy note:** Even with a strong channel key, MQTT metadata (sender node ID, timestamp, GPS coordinates if position sharing is enabled) may be visible to the broker operator and any subscriber on the same root topic. The public broker also enforces restrictions (zero-hop, location-precision limits, no broad `#` subscribe). Use a self-hosted broker for sensitive deployments.

Self-Hosted Broker Option

For deployments requiring privacy, control, or reliability guarantees, run your own MQTT broker (Eclipse Mosquitto is the standard choice). A self-hosted broker lets you:

- Require authentication (username + password or TLS client certificates).
- Restrict topic access with ACLs.
- Enable TLS on port 8883 with a Let's Encrypt or self-signed certificate.
- Bridge your private broker to the public one selectively.
- Log and retain messages for audit or replay.

MQTT Topic Hierarchy

Meshtastic publishes to a structured topic hierarchy:

```
msh/<region>/2/e/<channel_name>/!<node_id_hex>
```

Examples:

```
msh/US/2/e/LongFast/!abcd1234
msh/EU_868/2/e/MyCommunity/!1a2b3c4d
```

- `msh` - root topic for all Meshtastic traffic.
- `<region>` - geographic region prefix (US, EU_868, AU_915, etc.).

- `2/e` - protocol version 2, encrypted protobuf payload. (Firmware prior to 2.3.0 published to a topic with `/c/` in place of `/e/`.)
- `<channel_name>` - the channel name configured on the node (e.g. LongFast, MyCommunity).
- `!<node_id_hex>` - the 8-character hexadecimal user/node ID, preceded by `!`. This is the ID of the **gateway node that published the packet**, not necessarily the original sender - in a multi-hop mesh the originating sender and the publishing gateway can differ.

The payload is a binary protobuf `ServiceEnvelope` containing the `MeshPacket` (which may be encrypted). On the wire the protobuf bytes are binary, not base64 (base64 only appears in some viewer UIs like MQTT Explorer). Only devices that share the same channel key can decrypt the inner packet.

Uplink vs. Downlink

Direction	What it does	When to enable
Uplink	Sends LoRa packets heard locally to the MQTT broker	Must be enabled per channel (`uplink_enabled`), which defaults to off
Downlink	Receives packets from the broker and broadcasts them over LoRa	When you want the local mesh to hear remote traffic

Caution with downlink: enabling downlink on multiple gateways in the same RF coverage area causes duplicate transmissions and wasted air time. In a coverage area with more than one gateway, designate a single gateway for downlink and disable it on the others.

JSON vs. Protobuf Payload Formats

Meshtastic supports two payload formats on MQTT:

- **Protobuf (default)** - binary-encoded, compact, but requires a protobuf decoder to read. This is the format used by actual Meshtastic devices and by the public broker.
- **JSON** - human-readable, easier to consume by Node-RED, Home Assistant, scripts, etc. Published to a parallel topic `msh/<region>/2/json/<channel>/!<node_id>`. Enable **JSON Enabled** in the Meshtastic app under *Settings* > *MQTT*. Note: JSON output is only available on ESP32 gateways - nRF52-based devices (RAK4631 and other RAK/WisBlock boards) cannot publish JSON; use the protobuf topic with a decoder instead. Only certain packet types are serialized to JSON (TEXT_MESSAGE, TELEMETRY, NODEINFO, POSITION, WAYPOINT, NEIGHBORINFO, TRACEROUTE, DETECTION_SENSOR, PAXCOUNTER, REMOTE_HARDWARE), not all traffic.

JSON mode is ideal for integration work but produces larger payloads. Disable it on resource-constrained brokers with many active nodes.

Common Use Cases

- **Global channel bridging** - join your local LongFast mesh to the worldwide LongFast MQTT cloud.
- **Remote node monitoring** - a solar-powered node deep in the field uplinks its telemetry (battery voltage, environment sensor data) to MQTT; you query the broker from home.
- **Alert forwarding** - a Node-RED flow subscribes to MQTT, detects specific messages, and forwards them to Telegram or Discord.
- **Mesh analytics** - pipe MQTT JSON output into InfluxDB/Grafana for network coverage and message delivery statistics.

Revision #6
Created 2026-05-03 05:36:30 UTC by Mesh America Admin
Updated 2026-06-10 05:33:29 UTC by Mesh America Admin