MQTT & Internet Gateway

- Meshtastic MQTT Setup
- Building a Meshtastic Internet Gateway
- MQTT Topic Structure and Packet Format
- Preventing MQTT Message Loops

Meshtastic MQTT Setup

MQTT lets a Meshtastic node forward all mesh traffic to the internet, making your local mesh visible on the network map, bridging messages to internet clients, and enabling monitoring and logging. This is what feeds online community maps (e.g. via the MapReport packet) that support Meshtastic map reporting.

Warning — read before enabling: By default, Meshtastic publishes packets to the MQTT broker **UNENCRYPTED**, even on channels that use a PSK, unless you set `mqtt.encryption_enabled` to true. Enabling MQTT uplink on the default LongFast channel publishes that traffic to the public broker where **anyone can read it**, because the default `AQ==` key is publicly known. Node positions are exposed regardless of PSK. Only uplink channels whose content you intend to be public, and never assume a PSK alone protects content sent over MQTT. The downlink/inject risk on public channels is covered below.

How MQTT works in Meshtastic

When MQTT is enabled on a node:

1. Every mesh packet received by the node is forwarded to an MQTT broker over WiFi or TCP
2. The MQTT broker stores and redistributes the messages to other subscribers
3. The public Meshtastic MQTT broker (mqtt.meshtastic.org) shares filtered traffic — zero-hop only, a limited set of portnums, and reduced location precision — which feeds community maps. It is not an unrestricted public feed, but anything it does publish is visible to anyone subscribed.
4. Optionally, messages from the internet (MQTT) can be injected back into the local radio mesh

Hardware requirement: Direct MQTT requires a WiFi-capable device (ESP32-based: Heltec V3/V4, T-Beam, etc.). nRF52840 devices (T-Echo, T-Deck, RAK4631) have no onboard WiFi and cannot connect to MQTT over their own network, but they can use the **MQTT Client Proxy** to relay MQTT through a connected phone running the app (note: JSON output is not supported on nRF52).

Connecting to the public Meshtastic MQTT broker

Via the Meshtastic app

1. Go to **Settings → Module Config → MQTT**
2. Enable MQTT: toggle ON
3. MQTT Server Address: `mqtt.meshtastic.org`
4. Username: `meshdev`
5. Password: `large4cats`
6. TLS Enabled: toggle ON (recommended)
7. Map Reporting Enabled: toggle ON to publish a map report so your node appears on community maps that consume MapReport packets
8. Save

Note on the public channel: the default LongFast channel uses the publicly known `AQ==` key and has no real authentication. If you leave uplink (and especially downlink) enabled on it, anyone on the public broker can see your traffic, and — if downlink is on — inject messages into your local mesh. See the downlink and security notes below before enabling.

Via CLI

```
meshtastic --set mqtt.enabled true
meshtastic --set mqtt.address mqtt.meshtastic.org
meshtastic --set mqtt.username meshdev
meshtastic --set mqtt.password large4cats
meshtastic --set mqtt.tls_enabled true
meshtastic --set mqtt.map_reporting_enabled true
```

Map reporting publishes a MapReport packet. Current firmware also exposes related sub-settings (for example position precision and the publish interval) under the `mqtt.*` namespace; check the field names in your installed firmware version, as they have changed over time.

Channel settings for MQTT

MQTT is enabled per channel. By default, the primary channel (channel 0) is configured to uplink to MQTT. Verify that your channel has **Uplink Enabled** set to ON:

```
meshtastic --ch-index 0 --ch-set uplink_enabled true
```

Important: A PSK on your channel does *not* mean MQTT uploads are encrypted. By default the gateway decrypts packets and uplinks them **unencrypted** to the broker, even on channels with a

custom PSK. To keep message content encrypted on the broker you MUST set `mqtt.encryption_enabled = true`. Even with encryption enabled, packet metadata and node positions may still be exposed. With encryption enabled, community maps see only reduced-precision node positions and cannot read message content; the public server further filters location precision.

Downlink: receiving messages from the internet

Downlink allows messages published to MQTT to be injected into the local radio mesh - enabling internet-connected users to send messages that appear on mesh nodes in your area:

```
meshtastic --ch-index 0 --ch-set downlink_enabled true
```

Security note: Only enable downlink on channels with PSK authentication if you want to control who can inject messages into your local mesh. The public LongFast channel has no authentication - anyone on the public MQTT broker can inject messages into your mesh if downlink is enabled on the default channel.

Running a private MQTT broker

For a community or organizational network, run your own Mosquitto broker instead of using the public one:

```
# Install Mosquitto
sudo apt install mosquitto mosquitto-clients

# Basic config: /etc/mosquitto/mosquitto.conf
listener 1883 localhost # Local only (use nginx/TLS for external)
listener 8883 # TLS port for internet clients
cafile /path/to/ca.crt
certfile /path/to/server.crt
keyfile /path/to/server.key
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Point your Meshtastic nodes to your broker's address instead of mqtt.meshtastic.org.

MQTT topic structure

Meshtastic publishes to topics of the form:

```
msh/{region}/2/e/{channel_name}/{node_id}     # protobuf (encrypted/binary) topic
msh/{region}/2/json/{channel_name}/{node_id}  # JSON topic (when JSON output is enabled)

Examples:
msh/US/2/json/LongFast/!abcd1234
msh/US/2/e/LongFast/!abcd1234
```

Subscribe to `msh/US/#` to receive all US region traffic. The 4th segment is the **channel name**; the packet type is *not* a topic segment. On `msh/{region}/2/e/...` the payload is a protobuf-encoded `ServiceEnvelope`; a JSON-encoded payload appears only on `msh/{region}/2/json/...` when JSON output is enabled, and the packet kind (nodeinfo, position, text, etc.) is carried in the JSON `type` field inside the payload.

Building a Meshtastic Internet Gateway

A Meshtastic internet gateway bridges local LoRa radio traffic to the internet and can serve as a powerful community infrastructure node. This guide covers setting up a dedicated gateway on a Raspberry Pi.

Gateway hardware options

Option	Hardware	Pros	Cons
ESP32 node (simplest)	Heltec V3/V4, T-Beam	Single device, no Pi needed, compact	Single-channel, limited processing
Pi + LoRa hat	Raspberry Pi 4 + SX126x HAT	Full Linux environment via meshtasticd	More complex setup. Note: RAK2287/RAK5146 are LoRaWAN concentrators; Meshtastic on Linux (meshtasticd) uses a single-radio SX126x HAT, not a multi-channel LoRaWAN concentrator.
Pi + USB LoRa node	Raspberry Pi + RAK4631 USB	Easy setup, use standard Meshtastic firmware	Single channel only

Option 1: Dedicated ESP32 gateway node

The simplest approach: configure a Heltec V3/V4 or T-Beam as a dedicated gateway. This node doesn't need to be a router/repeater - its primary job is bridging radio and MQTT.

1. Flash with Meshtastic firmware (standard)
2. Connect to your home or community WiFi: `meshtastic --set network.wifi_enabled true --set network.wifi_ssid "YourSSID" --set network.wifi_psk "YourPassword"`
3. Configure MQTT as described in the MQTT Setup page

4. Leave the role at the default `CLIENT`. Do **not** set this gateway to ROUTER: on ESP32, ROUTER force-enables power-saving sleep and defaults WiFi/BLE/Serial OFF, which breaks a WiFi MQTT gateway. ROUTER is only for well-sited dedicated infrastructure radios, not for a node whose job is bridging to MQTT.
5. Mount at a good location with LoRa antenna and reliable WiFi

Privacy warning: A gateway that uplinks the default LongFast channel publishes *all* traffic it hears - including node positions - to the public internet, and (despite the channel PSK) Meshtastic uploads to MQTT **unencrypted by default** unless you set `mqtt.encryption_enabled true`. Use a private channel with encryption enabled, or accept that default-channel traffic is fully public. Set `lora.ignore_mqtt true` to prevent rebroadcast loops.

Option 2: Raspberry Pi + USB LoRa node

Connect an nRF52840-based device (RAK4631, T-Echo) via USB to a Raspberry Pi. The Pi handles internet connectivity while the LoRa node handles radio.

Setup the LoRa node

```
# Flash with Meshtastic firmware
# Leave the role at the default CLIENT for a USB-attached gateway radio.
# Do NOT set ROUTER, which disables serial and forces sleep on ESP32.
```

Install meshtasticd (Meshtastic daemon)

The Python CLI installs from PyPI, but `meshtasticd` is **not** on PyPI - install it from the official Meshtastic apt repository:

```
# Python CLI (PyPI)
pip3 install meshtastic

# meshtasticd daemon (apt repo, not pip)
sudo add-apt-repository ppa:meshtastic/beta
sudo apt update
sudo apt install meshtasticd
```

Configure MQTT bridging via Python

```
import meshtastic
import meshtastic.serial_interface
from pubsub import pub
import paho.mqtt.client as mqtt
import json

# Connect to the LoRa node
iface = meshtastic.serial_interface.SerialInterface("/dev/ttyUSB0")

# Connect to MQTT broker
mq = mqtt.Client()
mq.username_pw_set("meshdev", "large4cats")
mq.tls_set()
mq.connect("mqtt.meshtastic.org", 8883)

# Forward all received packets to MQTT.
# Topic format: msh/REGION/2/json/CHANNELNAME/USERID
# (version "2" comes before the channel name; packet type is a field inside the JSON, not a
topic segment)
def on_receive(packet, interface):
 userid = packet.get('fromId', '?')
 topic = f"msh/US/2/json/LongFast/{userid}"
 mq.publish(topic, json.dumps(packet))

# Packets are delivered via PyPubSub, not an iface.on_receive attribute
pub.subscribe(on_receive, "meshtastic.receive")

mq.loop_forever()
```

Note: `meshdev` / `large4cats` is the shared *public* broker, which is rate-limited, filtered, and not an open firehose. The firmware's built-in MQTT uplink is the supported, recommended path; this raw-publish script is illustrative only.

Monitoring your gateway

A healthy gateway should be publishing to MQTT continuously. Monitor with:

```
# Subscribe to your node's traffic (replace !abcd1234 with your node ID)
mosquitto_sub -h mqtt.meshtastic.org -p 8883 -t "msh/US/#" -u meshdev -P large4cats --tls-use-os-certs | grep abcd1234
```

You should see JSON packets appearing whenever your node hears a packet on the mesh. If the stream is silent for more than a few minutes in an active network, your WiFi or MQTT connection may have dropped.

Adding your gateway to the community map

To appear on third-party maps, your gateway needs MQTT uplink working plus map reporting enabled in the MQTT module config (`mapReportingEnabled`, firmware ≥2.3.2) and a fixed or GPS position set. Map reports publish at most about hourly, so allow up to an hour or more before your node shows up. Appearance on meshmap.net depends on that third-party site and is not instantaneous or guaranteed; verify by searching for your node name.

MQTT Topic Structure and Packet Format

Understanding Meshtastic's MQTT topic structure and packet encoding lets you build integrations, parse data, and troubleshoot gateway connectivity issues.

Topic Structure

Meshtastic publishes to MQTT topics using this hierarchy:

```
msh/{region}/2/{e|json}/{channel_name}/{user_id}

Examples:
msh/US/2/e/LongFast/!ab12cd34    # Encrypted packet, published by gateway !ab12cd34
msh/US/2/json/LongFast/!ab12cd34 # JSON-decoded packet (only if JSON enabled on the gateway)

Where:
 msh/      - Meshtastic prefix
 US/       - Region code (US, EU, etc.)
 2/        - Protocol version
 e/        - Encrypted protobuf (default)
 json/     - JSON decoded (optional, only if JSON output is enabled on the gateway)
 LongFast/ - Channel name (this segment is always present)
 !ab12cd34 - User/gateway node ID (the node that published this packet to MQTT, not
necessarily the originator)
```

The 4th segment is `e` (encrypted protobuf) or `json`; the 5th segment is the channel *name*; the last segment is the publishing (gateway) node's hex ID. The packet's *type* (text, telemetry, position, etc.) is a field inside the payload, not a topic segment - there are no `.../json/text/` or `.../json/telemetry/` topics.

Subscribing to All Traffic

```
# Subscribe to all Meshtastic packets from all US nodes:
mosquitto_sub -h mqtt.meshtastic.org -u meshdev -P large4cats -t "msh/US/2/e/#" -v

# Subscribe to a specific channel only:
mosquitto_sub -h mqtt.meshtastic.org -t "msh/US/2/e/CommunityMesh/#" -v

# Subscribe to JSON-decoded packets (if gateway has JSON enabled):
mosquitto_sub -h localhost -t "msh/US/2/json/#" -v
```

Packet Format

Each MQTT message payload is a protobuf-encoded `ServiceEnvelope` (defined in `meshtastic.protobuf.mqtt_pb2`) - the wrapper a gateway publishes - containing:

- **packet** - The MeshPacket (encrypted payload + routing info)
- **channel_id** - Channel name (e.g., "LongFast")
- **gateway_id** - Node ID of the gateway that published to MQTT

Enabling JSON Output on a Gateway Node

Security warning: JSON output publishes fully *decoded* (plaintext) packet content to MQTT, bypassing channel encryption entirely. The gateway can only emit JSON for packets it can decrypt (channels whose key it holds). Never enable `json_enabled` on a gateway that uplinks to a public or shared broker if any channel content is sensitive - anyone subscribed to that broker will read the cleartext.

```
# Enable JSON output (in addition to protobuf) on the GATEWAY node that publishes to MQTT.
# Only packets the gateway can decode (known channel key) are emitted as JSON:
meshtastic --set mqtt.json_enabled true

# JSON packets are published to: msh/REGION/2/json/CHANNELNAME/USERID
#   CHANNELNAME = the channel's name (e.g. LongFast)
#   USERID      = the gateway node's hex ID (e.g. !7efeee00)
# JSON payload example:
{
```

```
 "from": 2881537332,
 "to": 4294967295,
 "channel": 0,
 "type": "text",
 "id": 123456789,
 "rx_time": 1712000000,
 "hop_limit": 3,
 "payload": {
 "text": "Hello mesh!"
 }
}
```

Decoding Protobuf Packets in Python

```
import paho.mqtt.client as mqtt
from meshtastic.protobuf.mqtt_pb2 import ServiceEnvelope
from meshtastic.protobuf.portnums_pb2 import PortNum
import base64

def on_message(client, userdata, msg):
 try:
 se = ServiceEnvelope()
 se.ParseFromString(msg.payload)
 packet = se.packet
 print(f"From: !{packet.from_:08x}")
 print(f"To: !{packet.to:08x}")
 print(f"Channel: {se.channel_id}")
 # Decrypt and decode based on portnum for full payload
 except Exception as e:
 print(f"Parse error: {e}")

client = mqtt.Client()
client.username_pw_set("meshdev", "large4cats")
client.on_message = on_message
client.connect("mqtt.meshtastic.org", 1883)
```

```
client.subscribe("msh/US/2/e/#")
client.loop_forever()
```

Preventing MQTT Message Loops

A common misconfiguration in networks with MQTT gateways is the "MQTT loop" - packets sent over LoRa get forwarded to MQTT, which then re-injects them back into the LoRa network, causing each message to be transmitted multiple times and rapidly increasing channel utilization.

How MQTT Loops Happen

1. Node A sends a message over LoRa
2. Gateway node G receives the LoRa packet and publishes it to MQTT
3. MQTT broker delivers the packet to Gateway G's downlink subscription
4. Gateway G injects the packet back into the LoRa network
5. Node A receives its own message again as if from the MQTT cloud
6. This can loop indefinitely if not properly configured

Prevention: The ignore_mqtt Setting

A key anti-loop control is the `lora.ignore_mqtt` flag. When set to true, the device will ignore (and not re-broadcast) any messages it receives over LoRa that came via MQTT somewhere along the path toward the device. Note this only works when both your device and the MQTT node are running at least firmware version 2.2.19:

```
# Enable on ALL infrastructure nodes (routers, repeaters):
meshtastic --set lora.ignore_mqtt true

# Verify:
meshtastic --get lora.ignore_mqtt
```

Critical: Set this on every infrastructure/relaying node (ROUTER, ROUTER_LATE, and REPEATER roles). Leave it as false only on end-client nodes that may need to receive messages downlinked

from MQTT.

Gateway Configuration Best Practices

```
# Configure the gateway node correctly:

# 1. Enable MQTT publishing (uplink):
meshtastic --ch-index 0 --ch-set uplink_enabled true

# 2. Enable MQTT subscribing (downlink) ONLY if needed:
# Only enable downlink if you want to receive messages sent to MQTT
# from the global mesh or other external systems.
# If you only want to publish (monitoring), keep downlink disabled:
meshtastic --ch-index 0 --ch-set downlink_enabled false

# 3. Enable ignore_mqtt on the gateway node itself:
meshtastic --set lora.ignore_mqtt true
```

Detecting a Loop in Progress

Signs that a loop is occurring:

- Channel utilization (CU) rapidly increasing after enabling MQTT
- Messages appearing in the chat multiple times
- The same packet ID appearing in MQTT subscription twice or more within a few seconds
- Nodes reporting high packet counts in stats

```
# Check channel utilization. There is no --get for this; it is a live
# telemetry metric, read it from --info or the app's node metrics.
# Aim to keep it under ~25% (the firmware begins delaying transmissions above that):
meshtastic --info

# Monitor for duplicate packet IDs via MQTT:
mosquitto_sub -t "msh/US/2/e/#" -v | grep -E "packet.id"
# Watch for repeating IDs within 30 seconds
```