Monitoring and Maintenance

Remote monitoring stacks, firmware update procedures, and systematic troubleshooting for deployed Meshtastic repeaters.

Remote Monitoring a Meshtastic Repeater

A repeater deployed on a hilltop or building rooftop is useless to the community if failures go undetected for days. Effective remote monitoring lets you catch power issues, firmware hangs, and hardware faults before users notice. This page covers the monitoring stack - from MQTT uplink through time-series dashboards to alerting - and the procedures for restarting a stuck node remotely.

This MQTT/Telegraf/Grafana stack is an advanced, optional setup. If you just want a quick health check, the simple in-app telemetry view (battery, channel utilisation, last heard) in the Meshtastic app is the recommended starting point for most operators.

Important caveat - this monitoring path depends on internet connectivity at the repeater. The entire stack relies on the repeater reaching an MQTT broker over Wi-Fi/internet. During grid-down or internet-outage events - exactly the scenarios mesh emergency comms is built for - MQTT monitoring will be blind, and you lose visibility precisely when it matters most. Do not rely on MQTT/Grafana as your sole health check for incident readiness; maintain an over-the-mesh (LoRa) status check or a local/RF heartbeat that does not require internet.

Architecture Overview

The standard monitoring stack for a Meshtastic infrastructure node has four components:

1. **Node -> MQTT uplink**: The repeater connects to Wi-Fi (if ESP32-based) and publishes mesh packets as JSON to an MQTT broker. This requires internet at the repeater (see the caveat above).
2. **MQTT broker -> InfluxDB**: A subscriber (Telegraf or a custom script) consumes MQTT messages and writes telemetry fields into InfluxDB or another time-series database.
3. **Grafana -> InfluxDB**: Grafana dashboards visualise battery voltage, SNR of received packets, channel utilisation, and uptime over time.
4. **Alerting**: Grafana alerts (or a Python watchdog) send notifications when telemetry gaps indicate the node is offline.

Enabling the MQTT Uplink on the Repeater

```
# Enable MQTT on the device
meshtastic --set mqtt.enabled true

# Set your MQTT broker address
meshtastic --set mqtt.address mqtt.yourdomain.com

# Set MQTT username and password (if your broker requires auth)
meshtastic --set mqtt.username meshuser
meshtastic --set mqtt.password yourpassword

# Enable JSON encoding (required for Telegraf/InfluxDB ingestion)
meshtastic --set mqtt.json_enabled true

# Set the root MQTT topic (all messages published under this prefix)
meshtastic --set mqtt.root msh

# Enable uplink on the default channel (channel 0)
meshtastic --ch-set uplink_enabled true --ch-index 0
```

After applying these settings the node publishes JSON packets to topics of the form: `msh/US/2/json/<CHANNELNAME>/<USERID>` (for example `msh/US/2/json/LongFast/!abcd1234`). The path segments after `/json/` are the channel name and the node user ID - the portnum is not in the topic path, it appears inside the JSON message as the `type` field.

Telegraf Configuration for InfluxDB Ingestion

Install Telegraf on your monitoring server and add an MQTT consumer input:

```
[[inputs.mqtt_consumer]]
 servers = ["tcp://mqtt.yourdomain.com:1883"]
```

```
  topics = ["msh/#"]
  username = "meshuser"
  password = "yourpassword"
  data_format = "json"
  json_time_key = "rx_time"
  json_time_format = "unix"

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "YOUR_INFLUXDB_TOKEN"
  organization = "mesh-community"
  bucket = "meshtastic"
```

Restart Telegraf and verify data is flowing:

```
telegraf --config /etc/telegraf/telegraf.conf --test
```

Grafana Dashboard Panels

Create a Grafana dashboard with the following panels for each monitored repeater:

Battery Voltage Trend

```
from(bucket: "meshtastic")
  |> range(start: -7d)
  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")
  |> filter(fn: (r) => r["_field"] == "voltage")
  |> filter(fn: (r) => r["from"] == "!YOUR_NODE_ID")
```

Channel Utilisation Over Time

```
from(bucket: "meshtastic")
  |> range(start: -24h)
  |> filter(fn: (r) => r["_field"] == "channel_utilization")
  |> filter(fn: (r) => r["from"] == "!YOUR_NODE_ID")
```

Last Seen (Uptime Check)

Create a Stat panel showing the time since last telemetry packet. As a common rule of thumb (not a firmware spec), treat a node as likely offline once the time since last telemetry exceeds about 2× its telemetry broadcast interval.

Offline Detection: The 2× Interval Rule

Telemetry interval is a Module Config setting (under Telemetry in the app, not the radio LoRa config). It is broadcast on a configurable interval; set it on the repeater:

```
# Set device metrics telemetry interval to 30 minutes (1800 seconds)
meshtastic --set telemetry.device_update_interval 1800
```

Configure your monitoring system to alert if no telemetry has been received from the node in **2 × 1800 = 3600 seconds (1 hour)**. This tolerates a single missed packet (common with occasional MQTT delivery failures) before triggering an alert.

Be aware of the tradeoff: a 1-hour detection window means a failed node can be down for nearly an hour before you are alerted, and that failure could coincide with the start of an incident. Tune the window to how long you can tolerate an undetected outage, not just to MQTT reliability. For emergency infrastructure, shorten the telemetry interval (e.g. 5-10 minutes) so the 2× alert threshold gives 10-20 minute detection, or add an independent faster heartbeat.

In Grafana, create an Alert Rule on the Last Seen panel with the condition: *last seen > 3600 seconds -> ALERT*.

Python Watchdog with Telegram Alerts

For operators who prefer a lightweight Python watchdog over a full Grafana stack, the following script monitors MQTT and sends a Telegram message when a repeater goes silent.

```
#!/usr/bin/env python3
"""
Meshtastic repeater watchdog - sends Telegram alert when node goes silent.
Dependencies: pip install paho-mqtt requests
"""
import time, threading, paho.mqtt.client as mqtt, requests

MQTT_HOST = "mqtt.yourdomain.com"
MQTT_PORT = 1883
MQTT_USER = "meshuser"
MQTT_PASS = "yourpassword"
MQTT_TOPIC = "msh/#"

# Map node ID (string, e.g. "!abcd1234") to friendly name
WATCHED_NODES = {
 "!abcd1234": "Mt-Davidson Repeater",
 "!ef567890": "Twin-Peaks Repeater",
}

# Telegram bot config
TELEGRAM_TOKEN = "YOUR_BOT_TOKEN"
TELEGRAM_CHAT_ID = "YOUR_CHAT_ID"

# Alert if silent for longer than this many seconds
SILENCE_THRESHOLD = 3600 # 2x 30-minute telemetry interval

last_seen = {nid: time.time() for nid in WATCHED_NODES}
alerted = {nid: False for nid in WATCHED_NODES}

def send_telegram(msg):
 url = f"https://api.telegram.org/bot{TELEGRAM_TOKEN}/sendMessage"
 requests.post(url, data={"chat_id": TELEGRAM_CHAT_ID, "text": msg})

def on_message(client, userdata, msg):
 try:
 # Topic format is msh/US/2/json/CHANNELNAME/USERID, so the node USERID is the last segment
 parts = msg.topic.split("/")
 node_id = parts[-1]
 if node_id in last_seen:
 last_seen[node_id] = time.time()
```

```
 if alerted[node_id]:
 alerted[node_id] = False
 name = WATCHED_NODES[node_id]
 send_telegram(f"RESOLVED: {name} is back online.")
 except Exception:
 pass

def watchdog_loop():
 while True:
 now = time.time()
 for node_id, name in WATCHED_NODES.items():
 silent = now - last_seen[node_id]
 if silent > SILENCE_THRESHOLD and not alerted[node_id]:
 alerted[node_id] = True
 mins = int(silent / 60)
 send_telegram(
 f"ALERT: {name} ({node_id}) has been silent for {mins} minutes."
 )
 time.sleep(60)

client = mqtt.Client()
client.username_pw_set(MQTT_USER, MQTT_PASS)
client.on_message = on_message
client.connect(MQTT_HOST, MQTT_PORT)
client.subscribe(MQTT_TOPIC)

threading.Thread(target=watchdog_loop, daemon=True).start()
client.loop_forever()
```

Run this script as a systemd service on your monitoring server so it restarts automatically after reboots.

Checking Battery Voltage Trend for Solar Systems

Solar-powered repeaters require voltage trend analysis, not just instantaneous readings. A battery at 12.6 V at noon is fine; the same reading at 4 AM after a cloudy week indicates the system is not

fully recovering and may fail the following night.

In Grafana, create a panel showing battery voltage over the previous 7 days with a minimum threshold line at your low-voltage cut-off. Set this cut-off to your battery manufacturer's specified low-voltage-disconnect value rather than a single hardcoded number - commonly around 11.8-12.0 V for a 12 V lead-acid battery and roughly 3.0-3.3 V per cell for LiPo. Configure an alert if the daily minimum voltage is declining by more than 0.1 V per day.

Remote Administration via Admin Keys

When monitoring indicates a repeater has stopped responding but power is confirmed present, a firmware hang is the likely cause. Meshtastic supports remote administration, but note an important limitation: only `--set` and `--get` are supported over `--dest`. There is no supported remote `--reboot` (or `--reset`) command - `--reboot` is a local-only CLI action. To recover a remote node you can change its configuration via `--set` (or use the admin app); a true remote reboot is not available over the mesh.

```
# Read a setting from the target node remotely
meshtastic --dest '!abcd1234' --get device.role

# Change a setting on the target node remotely (only --set/--get are supported remotely)
meshtastic --dest '!abcd1234' --set device.role REPEATER
```

For this to work:

- On firmware 2.5 and later, remote administration uses public-key (PKC) admin keys: the remote repeater stores the controlling node's public key in one of its Admin Key fields (Security Config). The legacy shared admin-channel PSK is the method for pre-2.5 nodes only, and a 2.5+ node cannot be managed via the legacy shared-PSK admin channel.
- The monitoring node must be able to reach the repeater (directly or via mesh relay).
- The repeater must not be in a complete firmware hang - if the radio stack has crashed, even admin packets will not be processed. For that case, physical resilience measures are the fallback (see below).

Resilience against firmware hangs

Meshtastic has internal firmware watchdogs, but it exposes no user-configurable watchdog interval setting (there is no `device.watchdog_secs` field). If you need automatic recovery from a hard

firmware hang where even admin packets are not processed, rely on physical solutions instead: an external hardware watchdog timer that power-cycles the board, or a scheduled power-cycle (for example a timer or smart relay on the supply). These do not require any network connectivity to recover the node.

Firmware Updates on Deployed Repeaters

Meshtastic has no over-the-LoRa firmware push between mesh nodes - firmware cannot be sent from one node to another over the radio mesh. Firmware is updated via a USB connection to the device, or via Bluetooth OTA on supported nRF52 devices (e.g. RAK4631). For a repeater deployed on a rooftop, tower, or remote site, this generally means a site visit. Planning these maintenance windows carefully - and documenting configuration before you go - prevents accidental service outages and preserves network continuity for the community.

Over-the-Air Firmware Update: Not Over LoRa

It is worth stating clearly: there is no mechanism in Meshtastic firmware to push a firmware binary from one mesh node to another over LoRa. The "OTA" term in Meshtastic documentation means *Bluetooth OTA* - a supported firmware update path over BLE from a smartphone app for nRF52 devices, not a transfer over the LoRa mesh. BLE OTA is a documented, first-class update path on nRF52 boards (such as the RAK4631), not a stray term.

USB flashing is the most universally supported and reliable path across all platforms; nRF52 devices additionally support reliable BLE OTA, which is often the only practical path for a sealed enclosure. Plan accordingly when choosing repeater sites - a location that requires climbing a locked tower or a two-hour drive is a location you will want to update infrequently, so prioritise stability over bleeding-edge firmware on remote infrastructure nodes.

Planning a Maintenance Window

- **Notify the community** before updating a widely-used repeater. Post in your community channel or MQTT-linked group chat at least 24 hours in advance. Include the expected downtime window and the node name/ID.
- **Schedule during low-usage periods** - typically overnight or early morning on a weekday.

- **Check release notes** at `github.com/meshtastic/firmware/releases` before updating. Look for breaking changes in configuration format, channel encoding, or protocol version that could prevent the updated node from communicating with older clients.
- **Do not update all infrastructure nodes simultaneously**. Update one node, confirm it reconnects to the mesh and interoperates with other nodes, then proceed to the next.
- **If a site has only ONE repeater serving an area, treat its update as a planned outage.** Schedule it when the network is not needed, notify users that the path will be DOWN, and have a pre-flashed known-good spare or the prior firmware binary on hand to roll back on-site if the flash fails (a failed flash can leave the device in a crash loop). Never update a sole-path repeater remotely or without a rollback plan. Note that a "same-site second node" is not true redundancy against site power loss, lightning, or fire.

Exporting Configuration Before the Update

Configuration is not preserved across all firmware versions. Some updates change protobuf field IDs or introduce new mandatory fields, causing the stored configuration to load with default values after the firmware wipe. Always export before touching the device.

```
# Export current configuration to a YAML file
meshtastic --export-config > repeater_config_backup_$(date +%Y%m%d).yaml
```

This produces a YAML file with all channel configurations, device settings, LoRa settings, position, and module config. Store this file in a safe location - version control or your operations shared drive. (Note: `--info` output is *not* a restorable backup; only the `--export-config` YAML can be re-applied with `--configure`.)

Review the exported file before the update and note any values that are not visible in the app UI (e.g. custom channel PSK keys, non-default hop limits, MQTT credentials). These are the settings most likely to need manual restoration if automatic re-import fails.

Flashing Firmware via the Web Flasher

The Meshtastic web flasher at `flasher.meshtastic.org` is the recommended tool for **ESP32** hardware platforms (T-Beam, Heltec, etc.). It runs entirely in a Chromium-based browser and handles partition layout, bootloader, and firmware in a single operation. **nRF52 boards (RAK4631, T-Echo) are not flashed with the web serial flasher** - they update by drag-and-dropping a UF2 file onto the device's mass-storage drive after entering bootloader mode (double-tap reset), or via BLE OTA.

1. Connect the repeater hardware to your laptop via USB. Use a data-capable cable - power-only cables will not expose a serial port.
2. Navigate to `flasher.meshtastic.org` in **Chrome or Edge** (these are the officially supported browsers; other Chromium browsers such as Brave are based on the same engine and may work, but Web Serial support is not guaranteed).
3. Click **Flash Connected Device**. The browser will request access to the serial port - select the device from the list (typically `CP2102`, `CH340`, or `FTDI` depending on the board).
4. Select the correct hardware variant. Flashing the wrong variant can result in a non-booting device, so **always confirm the exact variant string against the live device dropdown on flasher.meshtastic.org before flashing** - variant labels can change between releases. Common examples (verify before use):
   - T-Beam v1.1 → `tbeam`
   - T-Beam v1.2 / AXP2101 → `tbeam-s3-core` (the AXP2101 PMU appears on more than one board; confirm your board's exact variant in the device list)
   - Heltec v3 → `heltec-v3`
   - RAK WisBlock 4631 → `rak4631` (flashed via UF2 drag-and-drop, not the web serial flasher)
5. Select the firmware version. For infrastructure nodes, prefer the latest stable release rather than alpha/nightly builds.
6. Click **Flash** and wait for the progress bar to complete. Do not disconnect the cable during flashing.

Alternative: CLI / esptool flashing

The web flasher (flasher.meshtastic.org) is the supported path for ESP32 devices. A command-line flashing route using `esptool` is documented in the official "Flashing Firmware" guide. The standalone `meshtastic-flasher` GUI project is legacy/unmaintained in favour of the web flasher; if you use any CLI tool, verify its current syntax against the official docs rather than relying on a fixed command line, as flags change between versions.

Restoring Configuration After the Update

```
# Restore a previously exported configuration (YAML) with --configure
# (there is no --import-config flag)
meshtastic --configure repeater_config_backup_20250101.yaml
```

After restoring, verify critical settings manually:

```
# Verify device role
meshtastic --get device.role

# Verify region (master legal control for band and power cap)
meshtastic --get lora.region

# Verify channel 0 PSK matches your network
meshtastic --info | grep -A 3 "channel_0"

# Verify MQTT settings
meshtastic --get mqtt

# Verify fixed position
meshtastic --get position

# Verify hop limit
meshtastic --get lora.hop_limit
```

Reboot the device after restoring configuration to ensure all settings take effect from a clean state:

```
meshtastic --reboot
```

Version Compatibility: What to Check Before Updating

Meshtastic nodes rebroadcast packets based on matching radio parameters even when they cannot fully decode newer packet types (managed flooding rebroadcasts on radio params, not on decode success). There is no formal version-negotiation handshake, so nodes running very different firmware versions may fail to decode some packet types from each other. Specific areas to check in release notes:

- **Protocol buffer schema changes** - new required fields in existing message types can cause older firmware to reject packets from a newly updated node.
- **Channel encryption format changes** - rare but occurred between some major version transitions. If your channel uses a custom PSK, test packet delivery after updating one node before updating others.
- **Modem preset changes** - some presets have been deprecated over time (e.g. `LONG_SLOW`, `VERY_LONG_SLOW`), but existing modem-preset enum IDs are stable and are not renumbered or reordered (`LONG_FAST` has always been ID 0). New presets are appended rather than reused, so a given preset name maps to the same setting across versions. Still verify the active modem preset by name after updating in case a default changed.

```
# Verify active modem preset after firmware update
meshtastic --get lora.modem_preset
```

Post-Update Validation Checklist

1. Device boots and appears in Meshtastic app node list.
2. Device role is correct. For infrastructure, use `ROUTER` (or `ROUTER_LATE` for nodes that should rebroadcast only after others). Note that `ROUTER_CLIENT` was deprecated in v2.3.15 and `REPEATER` is deprecated as of ~v2.7.11, so prefer `ROUTER` for new and updated deployments.
3. **LoRa region is set correctly for your location** (`meshtastic --get lora.region`). A firmware update that resets the region to UNSET or changes a default can cause out-of-band or wrong-power transmission - the region is the master legal control for both your authorized band and your power cap, so always confirm it after updating.
4. Channel 0 (primary) PSK matches community configuration.
5. Fixed position is present and correct on map.
6. MQTT uplink is publishing telemetry (check Grafana or broker).
7. Hop limit is correct for your network.
8. Position broadcast interval is set to a long static-node value (e.g. 43200 or 86400 seconds), not the default short interval.
9. At least one other mesh node can receive packets from the updated repeater.

Only after this checklist passes should you remove the USB cable and close up the enclosure.

Troubleshooting a Misbehaving Repeater

Infrastructure repeaters are expected to operate unattended for months. When behaviour deviates from normal - excessive channel utilisation, duplicate node entries, relay failures, or complete silence - rapid and systematic diagnosis avoids unnecessary site visits and minimises community impact. This page catalogues the most common symptoms, their likely causes, and the remediation steps including serial console diagnostics.

Symptom 1: High Channel Utilisation Caused by This Node

Diagnosis

In the Meshtastic app or web UI, open the node list and check the channel utilisation percentage. If it is above 25% and you suspect a specific node is the primary contributor, inspect each candidate node's reported airtime/channel-utilisation telemetry, or watch the serial/debug log of the suspect node to see how often it is transmitting and rebroadcasting. (Traceroute shows the routing path/hops to a destination - it does not measure which node rebroadcasts most frequently, so it is not the right tool for isolating a chatty node.) You can also check the MQTT stream for an unusually high publication rate from one node ID.

```
# On the suspect node: check current telemetry broadcast interval
meshtastic --get telemetry.device_update_interval
meshtastic --get telemetry.environment_update_interval

# Check position broadcast interval
meshtastic --get position.position_broadcast_secs
```

Common causes and fixes

Short telemetry intervals: If device_update_interval has been shortened well below the default, a node generates a telemetry packet every interval plus the corresponding rebroadcasts. (Note: a connected client app always receives device metrics once per minute regardless of this setting - that local behaviour is separate from the over-mesh broadcast interval, whose default is 1800 seconds / 30 minutes.) Set it back to 30 minutes for infrastructure nodes:

```
meshtastic --set telemetry.device_update_interval 1800
meshtastic --set telemetry.environment_update_interval 1800
```

Short position broadcast interval: A fixed repeater broadcasting position every 5 minutes generates 288 position packets per day. Increase to 24 hours:

```
meshtastic --set position.position_broadcast_secs 86400
```

Smart broadcast enabled on a non-moving node: Smart position broadcasting transmits whenever the node moves more than a configured distance. On a fixed node, GPS jitter can cause false movement detection and trigger frequent transmissions. Disable it:

```
meshtastic --set position.position_broadcast_smart_enabled false
```

Symptom 2: Node Appearing Twice in the Node List

Diagnosis

Two entries in the node list with the same name or similar names, but different node IDs, indicates that the same physical device has been flashed or configured with two different node identities. This typically happens when:

- A backup config from device A was imported onto device B, causing B to broadcast device A's NodeInfo until B's radio generates its own unique ID advertisement.
- A node was factory-reset, generating a new node ID, but other mesh nodes still have the old ID cached.
- Two devices were cloned by copying the flash image directly (not supported - never duplicate node configurations this way).

Remediation

If two physical devices have the same node ID (the critical case - never do this): One device must be factory-reset to generate a new unique ID. Factory reset clears all configuration including channel keys - re-configure from scratch:

Warning: A factory reset erases the channel PSK and all configuration - the node will fall off your mesh and emergency channel until it is manually re-keyed, which on a remote node means a site visit. NEVER factory-reset a sole-path or remote infrastructure node without (a) a current exported config backup (`meshtastic --export-config > config.yaml`) and (b) physical or admin access to restore it. Do not perform this during an active incident.

```
meshtastic --factory-reset
```

If stale entries persist after a legitimate device reset: Other mesh nodes cache node entries and expire stale ones on their own over time - there is no fixed published TTL, but the entry will age out without intervention. There is no remote command to force-clear another node's cache. Note that `--remove-position` only unsets a device's own fixed position (latitude/longitude/altitude) - it does NOT clear the node database. To reset a node's own NodeDB requires either the dedicated NodeDB-reset command or a factory reset; clearing a node's own DB does not remove a stale entry that other nodes are caching, which simply expires on its own.

```
# Clears only this device's own fixed position - NOT the node database:
meshtastic --remove-position
```

Symptom 3: Node Offline Despite Power Being Present

Diagnosis

The node is powered (LEDs lit, no alarm on power supply) but is not appearing in the mesh and is not publishing to MQTT. This indicates a firmware hang, a radio module fault, or a configuration issue that prevents the radio from transmitting.

Step 1: Attempt remote recovery

Remote administration over `--dest` supports only the `--set` and `--get` commands - there is no supported remote `--reboot` command, so a hung node generally cannot be rebooted over the air. You can remotely read or adjust settings if the radio stack is still responding:

```
# Remote admin supports only --get / --set over --dest:
meshtastic --dest '!abcd1234' --get device.role
```

If the radio stack is functional but the application layer is hung, remote reads may still respond. If the node is fully unresponsive, you will need a physical power cycle or serial console access (below). Meshtastic has no configurable hardware-watchdog setting to enable.

Step 2: Serial console inspection

Connect via USB and open a serial terminal at 115200 baud:

```
screen /dev/ttyUSB0 115200
# or on macOS:
screen /dev/cu.usbserial-0001 115200
# or using meshtastic CLI:
meshtastic --port /dev/ttyUSB0 --debug
```

Look for these patterns in the serial output (exact log wording varies by firmware version):

- `assert` or `Guru Meditation Error` - firmware crash, followed by register dump. Note the crash address for reporting to the Meshtastic GitHub issues.
- Messages indicating no packets are being received - the radio module may be locked up. Try power cycling.
- `nvs_flash` errors - NVS (non-volatile storage) corruption. Full flash erase followed by re-flash and reconfiguration is required.
- Looping reboot messages without the firmware reaching a completed-boot state - this can indicate a hardware fault; one possible cause is a damaged SPI bus between the ESP32 and the LoRa module.

Step 3: Full power cycle

Disconnect power completely (including USB) for 10 seconds. Many transient LoRa module hang states clear only with a full power cycle, not a software reset.

Symptom 4: Node Not Relaying Packets

Diagnosis

The node appears in the node list and is visible on the map, but traceroutes confirm it is not serving as a relay hop - packets that should transit through it are not being forwarded.

Check the device role

A node configured as `CLIENT_MUTE` does not relay packets from other nodes. A plain `CLIENT` node DOES relay - it rebroadcasts packets when no other node has already done so - so a CLIENT role is not, by itself, the reason a node fails to forward. Confirm the role:

```
meshtastic --get device.role
```

If the role returns `CLIENT_MUTE` (which suppresses rebroadcasting), change it. For a well-placed stationary infrastructure node, `ROUTER` is the recommended role:

```
meshtastic --set device.role ROUTER
```

Check the rebroadcast mode

```
meshtastic --get device.rebroadcast_mode
```

The rebroadcast mode is set under the `device.` namespace (`device.rebroadcast_mode`). The valid values are:

- `ALL` - rebroadcast all packets (the default).
- `ALL_SKIP_DECODING` - same as ALL but skips packet decoding and simply rebroadcasts. This only takes effect on the **REPEATER** role; it is not a general relay mode for other roles.
- `LOCAL_ONLY` - ignore packets from foreign meshes (other regions/modem presets), relaying only local-mesh traffic.
- `KNOWN_ONLY` - relay only packets from nodes already in the NodeDB, filtering out unknown/foreign nodes.
- `NONE` - do not rebroadcast (only valid for SENSOR/TRACKER/TAK_TRACKER roles).
- `CORE_PORTNUMS_ONLY` - rebroadcast only core protocol port numbers.

A mode of `LOCAL_ONLY` or `KNOWN_ONLY` filters foreign-mesh traffic and may explain why some transit traffic is being dropped.

Check hop count on arriving packets

If packets arrive at this node with a remaining hop count of 0, they are consumed and not forwarded - this is correct behaviour, not a fault. Verify by enabling debug output and watching the hop_limit field in received packets:

```
meshtastic --debug 2>&1 | grep "hop_limit"
```

If all arriving packets have hop_limit = 0, the originating nodes may need their `lora.hop_limit` increased (default 3, maximum 7). Raising the hop limit increases airtime and can worsen congestion, so do this cautiously and watch channel utilisation.

Check channel configuration

A relay forwards packets based on the unencrypted packet header (sender NodeID + packet ID), and it deduplicates using that same header - so a node can and does rebroadcast packets even when it does not hold the channel PSK. This is exactly why `ALL_SKIP_DECODING` can relay channels it cannot decrypt. A PSK mismatch prevents reading the message content, not relaying it. To actually exchange and read messages with the nodes you are serving, the channel 0 PSK must match:

```
meshtastic --info | grep -A 5 "channel_0"
```

Serial Console Diagnostic Reference

When physically on-site, the following serial console commands provide rapid triage:

```
# Full device info dump (version, channels, config, node DB)
meshtastic --info

# Export the full current configuration to a YAML backup file
meshtastic --export-config > config.yaml

# Watch live packet events (requires --debug flag)
meshtastic --debug

# Factory reset (last resort - wipes config and channel keys; back up first)
meshtastic --factory-reset
```

Reading the debug log for relay events

When `--debug` is active, relay events appear in the log. The exact wording varies by firmware version, but conceptually you will see rebroadcast events (the node forwarding a packet with a decremented hop limit) and duplicate-suppression events (the node ignoring a packet it has already heard), for example:

```
(illustrative, not verbatim)
Rebroadcasting packet from !abcd1234 ...
Ignoring duplicate packet from !abcd1234 (id=0x12345678)
```

If you see only duplicate-suppression lines and no rebroadcast lines, the node is receiving packets but treating them all as duplicates. This can happen if the duplicate detection window (stored in RAM) has stale entries from a previous session - a reboot typically clears this.

If neither message type appears for packets that other nearby nodes are hearing, the LoRa receive path is not functioning. Check antenna connection (a disconnected antenna is the single most common field repair) and SPI bus integrity.