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.

Revision #3
Created 2026-05-03 05:50:07 UTC by Mesh America Admin
Updated 2026-06-09 00:28:28 UTC by Mesh America Admin