Building a Mesh Network Dashboard

A community mesh network dashboard gives operators a real-time view of network health - which nodes are online, battery levels, channel utilization, and connectivity maps. This page covers building a monitoring stack for a Meshtastic network.

Architecture Overview

The standard monitoring stack for Meshtastic:

```
Meshtastic nodes
 ↓ (MQTT uplink)
MQTT Broker (Mosquitto)
 ↓
Telegraf (or Python consumer)
 ↓
InfluxDB (time-series database)
 ↓
Grafana (dashboard and alerting)
```

This stack can run on a Raspberry Pi 4 or any small Linux server. As a rough guide, budget at least 1 - 2 GB RAM (InfluxDB 2.x is memory-hungry and can exceed 500 MB under load) and on the order of 10 GB of storage per month of retained data for a ~50-node network. Actual usage varies with telemetry rate and retention - test on your own hardware before committing to a Pi.

Step 1: MQTT Broker Setup

Install Mosquitto on your monitoring server:

```
sudo apt install mosquitto mosquitto-clients
sudo systemctl enable mosquitto
```

Minimal config at `/etc/mosquitto/mosquitto.conf`:

```
listener 1883
allow_anonymous true
persistence true
persistence_location /var/lib/mosquitto/
```

For production, add TLS and password authentication.

Step 2: Configure Nodes to Uplink

On each node you want to monitor. Note that `uplink_enabled` is a per-channel setting, not an `mqtt.*` module key - there is no `mqtt.uplink_enabled`. Set the module-level keys first, then enable uplink on the channel:

```
# Module-level MQTT settings
meshtastic --set mqtt.enabled true
meshtastic --set mqtt.address "YOUR_SERVER_IP"
meshtastic --set mqtt.json_enabled true

# Per-channel: enable uplink on channel index 0 (repeat for other channels)
meshtastic --ch-index 0 --ch-set uplink_enabled true
```

JSON mode outputs human-readable JSON instead of protobuf binary - easier for custom consumers but includes less data. For full data including all telemetry, use protobuf mode and decode with the meshtastic Python library.

Step 3: InfluxDB

Install InfluxDB 2.x via the official InfluxData apt repository (the snippet below is abbreviated - follow the full add-key + add-repo steps in the InfluxData install guide at docs.influxdata.com/influxdb/v2/install/ before running `apt install`):

```
# Abbreviated - see docs.influxdata.com/influxdb/v2/install/ for the
# complete key import and repository setup steps
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
# ... import the key and add the influxdata apt source, then:
sudo apt install influxdb2
```

```
sudo systemctl enable --now influxdb
```

Create a bucket named "meshtastic" via the InfluxDB UI at http://localhost:8086. Set the retention period to match your storage budget - the ~10 GB/month figure above assumes roughly 30-day retention, so a 90-day retention would store about three times as much.

Step 4: Python MQTT-to-InfluxDB Bridge

Note: the bridge below subscribes to `msh/+/2/json/#`, the JSON subtopic, so that `json.loads()` only ever sees JSON payloads. If you instead subscribe to `msh/#` you will also receive raw protobuf topics, and `json.loads()` will throw on those binary payloads - the `try/except` simply skips anything that is not valid JSON.

```
pip install paho-mqtt influxdb-client meshtastic

# bridge.py - subscribe to Meshtastic MQTT, write metrics to InfluxDB
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import json, time

INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN = "YOUR_TOKEN"
INFLUX_ORG = "meshamerica"
INFLUX_BUCKET = "meshtastic"

client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=INFLUX_ORG)
write_api = client.write_api(write_options=SYNCHRONOUS)

def on_message(mqttc, userdata, msg):
 try:
 data = json.loads(msg.payload)
 except (ValueError, UnicodeDecodeError):
 # Non-JSON (raw protobuf) payload - skip it
 return
 try:
 node_id = data.get("from", "unknown")
```

```
        if "payload" in data:
            p = data["payload"]
            pt = Point("node_telemetry").tag("node_id", node_id)
            if "battery_level" in p:
                pt = pt.field("battery_pct", p["battery_level"])
            if "voltage" in p:
                pt = pt.field("voltage", p["voltage"])
            if "channel_utilization" in p:
                pt = pt.field("channel_util", p["channel_utilization"])
            write_api.write(INFLUX_BUCKET, record=pt)
    except Exception as e:
        print(f"Error: {e}")

mqttc = mqtt.Client()
mqttc.on_message = on_message
mqttc.connect("localhost", 1883)
# Subscribe only to the JSON subtopic so json.loads never sees protobuf
mqttc.subscribe("msh/+/2/json/#")
mqttc.loop_forever()
```

The telemetry JSON keys used above (`battery_level`, `voltage`, `channel_utilization`) correspond to the device-metrics telemetry fields; verify the exact key names and nesting against a live TELEMETRY_APP JSON sample from your firmware version, as the serializer can change.

Step 5: Grafana Dashboard

Install Grafana and add InfluxDB as a data source. Useful panels:

- **Battery Voltage by Node** - Time series, grouped by node_id tag
- **Online Node Count** - Count distinct nodes seen in last 30 minutes
- **Channel Utilization Heatmap** - Spot congestion patterns over time
- **Last Seen Table** - Node name and minutes since last packet

Alerting

Configure Grafana alerts to notify via email, Slack, or Telegram:

- Battery below 20% - node needs attention
- Node offline (no data in 2 hours) - repeater may be down
- Channel utilization above 30% - network congestion warning

Revision #3
Created 2026-05-03 05:51:13 UTC by Mesh America Admin
Updated 2026-06-09 22:22:48 UTC by Mesh America Admin