Monitoring Battery State via Meshtastic Telemetry

Meshtastic and MeshCore both include power telemetry features that allow a node to report its battery voltage and charge level over the mesh network. This page covers enabling these features, configuring voltage ADC pins for different hardware, interpreting voltage as state-of-charge for LiFePO4 batteries, setting low-battery alerts, and visualising data in Grafana. (Firmware command syntax and pin details are as of 2026-06-08; verify against your installed firmware version, as Meshtastic config keys change across 2.x releases.)

Enabling Power Telemetry in Meshtastic

In Meshtastic firmware (2.x), power metrics are part of the **Telemetry Module**. To enable battery reporting:

Via Meshtastic Python CLI

```
# Install CLI: pip install meshtastic
# Set the device-metrics broadcast interval (battery level, voltage, uptime,
# and any attached power sensor). This specifically controls the DEVICE/power
# telemetry cadence, not environment-sensor metrics.
meshtastic --set telemetry.device_update_interval 300
# Sets reporting interval to 300 seconds (5 minutes)

# Verify telemetry module settings
meshtastic --get telemetry
```

Note: per the official Meshtastic telemetry docs, `device_update_interval` configures the interval (in seconds) used to send device/power metrics over the mesh. CLI key names have changed across firmware versions — if the command above is rejected, check the current key names in the Meshtastic telemetry module docs for your firmware version.

Via Meshtastic Web App or Mobile App

1. Open the Meshtastic app and connect to your node.
2. Navigate to **Config → Module Config → Telemetry**.
3. Enable **Device Metrics**.
4. Set the update interval (300 - 3600 seconds; use longer intervals for battery-powered nodes to reduce TX duty cycle).
5. Save and reboot the node.

Once enabled, the node broadcasts a `meshtastic.Telemetry` protobuf packet on the default channel at the configured interval. The packet includes:

- `battery_level`: Integer 0 - 100 (firmware-estimated SoC percentage)
- `voltage`: Float in volts (actual measured ADC voltage)
- `channel_utilization`: Float (% airtime used)
- `air_util_tx`: Float (TX airtime)

Voltage ADC Pin Configuration on Different Boards

Not all Meshtastic hardware platforms use the same pin or divider ratio for battery voltage measurement. The firmware auto-detects the board type from compile-time defines, but custom builds or off-label hardware may need manual configuration. Verify the exact pin and divider against your board's schematic before relying on these values.

Board	ADC Pin (GPIO)	Voltage Divider Ratio	Max Measurable Voltage	Notes
TTGO T-Beam v0.7	GPIO35	1:2 (100 kΩ / 100 kΩ)	~8.4 V	Measures raw single-18650 LiPo voltage (divider documented in community schematics)
TTGO T-Beam v1.1 (AXP192)	AXP192 PMIC register	Internal PMIC ADC	Reported via I²C	Reads VBAT register; very accurate
TTGO LoRa32 v2.1	GPIO35	1:2	~8.4 V	Same GPIO35 divider scheme as T-Beam v0.7 (confirm on the v2.1 schematic)

Board	ADC Pin (GPIO)	Voltage Divider Ratio	Max Measurable Voltage	Notes
Heltec WiFi LoRa 32 v3	GPIO1 (VBAT_Read)	~4.9:1 divider (390 kΩ / 100 kΩ)	Single-cell LiPo (~3.0–4.2 V)	You must drive **ADC_Ctrl (GPIO37) low first** to enable the divider, then read battery voltage on GPIO1. It is **not** a 1:1 / no-divider input. Reads a single LiPo cell.
RAK WisBlock RAK4631	P0.04 (AIN2)	1:2 via RAK5005-O base board	~6 V	Battery sense is on P0.04 / AIN2 (not P0.05/AIN3); reads via nRF52840 SAADC. Verify the AIN index against the WisBlock schematic.
Wispr / Custom ESP32	User-defined GPIO	User-defined	User-defined	Set in platformio.ini or via power.adc_multiplier_override config key

If the reported voltage seems incorrect, verify with a multimeter at the battery terminals. Then check `power.adc_multiplier_override` (confirm this key exists in your firmware version):

```
meshtastic --set power.adc_multiplier_override 2.0
# Multiplies the raw ADC reading by 2.0 (use for 1:2 divider boards)
```

Interpreting Voltage as State-of-Charge for LiFePO4

Meshtastic's built-in SoC estimation uses LiPo voltage thresholds (3.0 - 4.2 V per cell). For LiFePO4 packs, these thresholds are incorrect - LiFePO4 cells operate in the 2.5 - 3.65 V range. The firmware will report incorrect percentages unless you compensate.

Important: A LiFePO4 cell's voltage during/just after charging is much higher than its *rested* open-circuit voltage. The 3.60–3.65 V "100%" row below is the charge/absorb voltage; a fully charged cell that has rested settles to roughly **3.35–3.45 V**. Read SoC from rested voltage, not from voltage measured under charge or load.

LiFePO4 Single-Cell (3.2 V nominal) Voltage → SoC Table

Voltage (V)	Approximate SoC (%)	Interpretation
3.60 - 3.65 (under charge)	100%	Charge/absorb voltage; rested 100% OCV is ~3.35–3.45 V
3.40 - 3.45	90%	High charge, near rested-full plateau
3.30 - 3.35	70 - 80%	Mid-range - most of usable capacity here
3.27 - 3.30	50%	Flat region - voltage barely distinguishable from 70%
3.22 - 3.25	30%	Still flat; lower usable threshold approaching
3.18 - 3.22	20%	Low battery - alert threshold
3.10 - 3.18	10%	Critical - immediate recharge needed
< 3.10	<5%	BMS will soon disconnect; node will shut down

4S LiFePO4 Pack (12.8 V nominal) Voltage → SoC Table

Pack Voltage (V)	SoC (%)
14.2 - 14.6 (under charge)	100% (end of charge; rested-full ~13.4–13.8 V)
13.6 - 13.8	90%
13.2 - 13.4	70 - 80%
13.0 - 13.2	50%
12.8 - 13.0	30%
12.4 - 12.8	20%
12.0 - 12.4	10%
< 11.8	<5% (BMS cutoff imminent)

Setting Low-Battery Alerts in Meshtastic

Meshtastic does not natively send alert messages when battery drops below a threshold, but there are two approaches to implement this:

Approach 1 - Node-Red / MQTT Alert Pipeline

1. Configure Meshtastic MQTT uplink: `meshtastic --set mqtt.enabled true --set mqtt.address YOUR_BROKER_IP`
2. In Node-Red, subscribe to `msh/US/+/json/LongFast/#` (adjust channel name as needed).
3. When Meshtastic JSON-encoded MQTT output is enabled, battery voltage arrives as a **flat field** inside the telemetry payload, e.g. `msg.payload.payload.voltage` (the published JSON looks like `"payload": {"air_util_tx":0, "battery_level":0, "channel_utilization":0, "voltage":0}`). It is **not** nested under `decoded.telemetry.deviceMetrics` — that dotted path is the protobuf-decoded object shape used by the Python API, not the MQTT JSON shape. Filtering on the wrong layer returns `undefined` and the alert silently never fires. Check a real sample against the Meshtastic MQTT integration docs and filter on the field below your LVD threshold.
4. Route low-voltage events to an alert node (email, PushOver, Telegram bot).

Approach 2 - Meshtastic Python Script (Autonomous Node)

```
import meshtastic
import meshtastic.serial_interface
from meshtastic.mesh_pb2 import MeshPacket

iface = meshtastic.serial_interface.SerialInterface()
LOW_VOLTAGE_THRESHOLD = 3.18 # V per cell for LiFePO4 (20% SoC)

def on_receive(packet, interface):
 if "decoded" in packet and "telemetry" in packet["decoded"]:
 m = packet["decoded"]["telemetry"].get("deviceMetrics", {})
 voltage = m.get("voltage", 0)
 node_id = packet["fromId"]
```

```
 if voltage > 0 and voltage < LOW_VOLTAGE_THRESHOLD:
 print(f"LOW BATTERY: Node {node_id} at {voltage:.2f} V")
 # Send alert message on mesh
 iface.sendText(f"⚠ Low battery: {node_id} {voltage:.2f}V", wantAck=False)

iface.localNode.setOwner("MonitorNode")
iface.addReceiveObserver(on_receive)
input("Press Enter to exit\n")
iface.close()
```

MeshCore Telemetry Equivalent

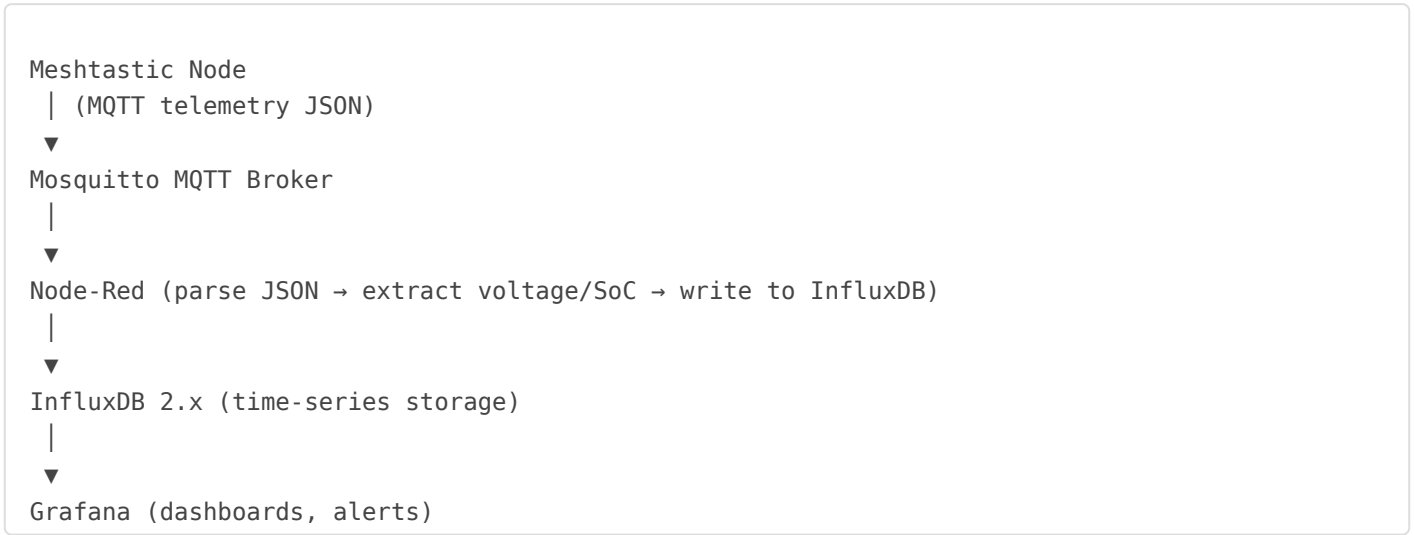
MeshCore also reports battery telemetry, but it does **not** use a "node YAML configuration file." MeshCore repeater and room-server firmware is configured over USB in the **web config tool**, or remotely over LoRa via the **companion mobile app** using the Remote Management feature, and via the **MeshCore CLI commands**. There is no YAML config mechanism in MeshCore firmware.

Use the MeshCore Repeater & Room Server CLI Reference to set the telemetry/advert interval and review available battery-reporting commands for your firmware build.

When telemetry is bridged to MQTT, Grafana can consume it via the Grafana MQTT data source plugin or via InfluxDB (Node-Red → InfluxDB → Grafana).

Graphing Battery Data in Grafana

Architecture



InfluxDB Line Protocol (Node-Red write node)

```
measurement: node_battery
tags: node_id, node_name, location
fields: voltage (float), battery_pct (int), soc_lifepo4 (float)
timestamp: nanosecond UNIX timestamp from packet
```

Grafana Panel Configuration

- **Battery voltage time series:** Use a Time Series panel with threshold bands - green above 3.30 V, yellow 3.18 - 3.30 V, red below 3.18 V (per cell) or scale for your pack voltage.
- **Multi-node SoC gauge:** Use a Gauge panel per node with min=0, max=100, thresholds at 20% (red) and 40% (yellow).
- **Grafana Alerting:** Set an alert rule on `avg(voltage) < 3.18` for any node, with a 15-minute evaluation window (to avoid false alarms from momentary load spikes). Route to PagerDuty, Slack, or email.

A reference Grafana dashboard for Meshtastic power monitoring can be built from the panel configuration above. (A previous version of this page referenced a specific dashboard JSON file in the Mesh America GitHub repository; that path is not yet verified to exist, so treat it as planned rather than published — build your dashboard from the configuration described here until a confirmed repository link is available.)

Revision #6
Created 2026-05-03 05:39:22 UTC by Mesh America Admin
Updated 2026-06-08 22:46:20 UTC by Mesh America Admin