

Meshtastic Repeaters

How to plan, configure, and deploy Meshtastic repeaters to extend network coverage.

- [☐ Start Here — Meshtastic Repeaters Guide](#)
- [Overview](#)
 - [What is a Meshtastic Repeater?](#)
 - [Device Roles Explained](#)
- [Site Planning](#)
 - [Choosing a Repeater Location](#)
 - [Antenna and Signal Range Factors](#)
 - [Using the Meshtastic Network Map](#)
- [Hardware](#)
 - [DIY vs Pre-built Kits](#)
 - [Power Systems for Repeaters](#)
- [Configuration](#)
 - [Setting the Device Role](#)
 - [Rebroadcast Mode: ALL_SKIP_DECODING](#)
 - [Power Optimization Settings](#)
 - [Choosing a Modem Preset](#)
- [Best Practices](#)
 - [Deployment Best Practices](#)
 - [Legal Considerations](#)

- [Backing Up Your Configuration](#)
- [Network Troubleshooting](#)
 - [Diagnosing Meshtastic Network Problems](#)
 - [Repeater Performance and Maintenance](#)
- [Meshtastic Repeater Setup](#)
 - [Router vs. Repeater Role — Which to Choose](#)
 - [Initial Setup Walkthrough](#)
 - [Channel Configuration for Infrastructure Nodes](#)
- [Advanced Meshtastic Repeater Topics](#)
 - [Store and Forward](#)
 - [Position and Telemetry for Infrastructure Nodes](#)
- [Role Configuration and Tuning](#)
 - [ROUTER vs ROUTER_CLIENT vs REPEATER: When to Use Each](#)
- [Role Configuration and Tuning](#)
 - [ROUTER vs ROUTER_CLIENT vs REPEATER Role: Deep Dive](#)
 - [Hop Limit Configuration for Repeaters](#)
 - [Fixed Position for Repeater Nodes](#)
- [Monitoring and Maintenance](#)
 - [Remote Monitoring a Meshtastic Repeater](#)
 - [Firmware Updates on Deployed Repeaters](#)
 - [Troubleshooting a Misbehaving Repeater](#)
- [Meshtastic Repeater Network Patterns](#)
 - [Designing for Reliability: N+1 Redundancy](#)
 - [Planned Maintenance Procedures for Live Networks](#)
 - [Channel Utilization Management](#)

☐ Start Here — Meshtastic Repeaters Guide

This book covers Meshtastic infrastructure nodes - the ROUTER and REPEATER roles (and ROUTER_LATE) - from site selection through deployment, configuration, monitoring, and maintenance. (Note: the older ROUTER_CLIENT role was deprecated and removed in firmware 2.3.15 — use ROUTER or ROUTER_LATE instead. As of firmware ~2.7.x the REPEATER role is also deprecated; for most infrastructure the current recommendation is ROUTER. See the role pages for details.)

☐ Where to Start

1. [What is a Meshtastic Repeater?](#)
2. [ROUTER vs REPEATER vs ROUTER_LATE: When to Use Each](#) - Critical decision before you configure anything. This is the canonical role-decision page; start here for which role to pick.
3. [Choosing a Repeater Location](#)

A note on the role pages: several pages in this book discuss role selection (Device Roles Explained, the Deep Dive, and Router vs. Repeater: Which to Choose). Treat "ROUTER vs REPEATER vs ROUTER_LATE: When to Use Each" above as the authoritative, current-firmware reference; the other role pages provide background but predate the ROUTER_CLIENT removal (2.3.15) and REPEATER deprecation (~2.7.x), so where they disagree, follow the canonical page.

☐ What's In This Book

Roles and Role Configuration

- [Device Roles Explained](#)
- [Router & Repeater Roles: Deep Dive](#)
- [Rebroadcast Mode: ALL_SKIP_DECODING](#)

- [Hop Limit Configuration for Repeaters](#)
- [Fixed Position for Repeater Nodes](#)

Site Planning and Setup

- [Choosing a Repeater Location](#)
- [Antenna and Signal Range Factors](#)
- [Initial Setup Walkthrough](#)
- [Channel Configuration for Infrastructure Nodes](#)
- [Legal Considerations](#) - FCC power and antenna rules before you transmit

Advanced Features

- [Store and Forward](#) - Message persistence for offline nodes
- [Position and Telemetry for Infrastructure Nodes](#)

Network Reliability and Redundancy

- [Designing for Reliability: N+1 Redundancy](#)
- [Planned Maintenance Procedures for Live Networks](#)
- [Channel Utilization Management](#)

Monitoring and Maintenance

- [Remote Monitoring a Meshtastic Repeater](#)
- [Firmware Updates on Deployed Repeaters](#)
- [Troubleshooting a Misbehaving Repeater](#)

➔ Related Books

- [Meshtastic](#) - Protocol, configuration, and app guide
- [Room Servers & Gateways](#) - MQTT gateways
- [Solar & Power Systems](#) - Powering outdoor repeaters

- [Network Planning](#) - Coverage planning and topology design

Overview

What repeaters are and the different device roles available.

What is a Meshtastic Repeater?

A Meshtastic repeater is a dedicated node whose purpose is to receive and retransmit messages, extending the reach of the mesh network beyond what any single device can achieve on its own.

In a Meshtastic network, every node participates in relaying messages to some degree - this is the nature of a mesh. But dedicated repeater nodes are optimized specifically for relaying: they run continuously and are placed at elevation for maximum coverage. Depending on the role chosen, an infrastructure node can either rebroadcast with higher priority (without deferring to neighbors) or be configured to stay silent on the mesh. Note that "repeater" is used here as a general concept: in practice these nodes use either the **ROUTER** role (which still broadcasts its own NodeInfo and position) or the **REPEATER** role (which relays silently without broadcasting its own data). These roles are covered in detail on the device-role pages.

Why repeaters matter

In real-world conditions, LoRa range is often limited by terrain, buildings, and vegetation. Range figures are highly variable and depend on preset, antenna, height, TX power, and obstructions. With stock antennas at Long Fast, urban non-line-of-sight range is often well under 1 km up to a few km; rural line-of-sight can be tens of km, and elevated sites with clear line-of-sight have achieved 100+ km. A repeater at elevation with a clear view can bridge gaps that would otherwise isolate parts of the network.

The Meshtastic relay model

Meshtastic uses a **managed flooding** approach: when a message is sent, nearby nodes rebroadcast it, and those nodes rebroadcast to others. A hop counter (default 3, maximum 7) limits how many times a message can be relayed before it is discarded. Dedicated repeaters - placed at strategic high points - maximize the effective reach of each hop.

Device Roles Explained

Meshtastic devices can be assigned one of several roles that control how they participate in the mesh. Choosing the right role for your repeater affects network behavior, power consumption, and visibility.

CLIENT (default)

The standard role for personal devices, and the role official Meshtastic guidance recommends for the overwhelming majority of ordinary nodes. A CLIENT node sends and receives messages via the app and participates in relaying (managed flooding): it rebroadcasts a packet only when no other node has already done so. If a CLIENT hears another node rebroadcast a packet first, it defers and will not rebroadcast it itself.

A CLIENT also ignores any packet (tracked by Packet ID) it has already received, so a given packet is never relayed twice by the same node. This managed-flooding behavior is what prevents broadcast storms.

Best for: personal devices carried by users, and most ordinary stationary nodes.

REPEATER

A dedicated relay node. **REPEATER is deprecated as of firmware ~2.7.x and is no longer recommended as the default for new infrastructure** (see "Which role should I choose?" below). Key behaviors:

- Rebroadcasts each valid packet once, with higher priority - it does not defer to other nodes the way a CLIENT does - subject to channel-activity detection (it still waits for the channel to be clear)
- **Does not appear in the node list** - it sends no NodeInfo, so it is anonymous on the mesh and keeps the network list clean
- Does not broadcast its own GPS or node info (saves network bandwidth)
- Does not force power-saving sleep (its radio stays active)
- Can be paired with the `ALL_SKIP_DECODING` rebroadcast mode for maximum efficiency (see below)

Note: `ALL_SKIP_DECODING` is a `device.rebroadcast_mode` value - a separate setting from the device role - that relays packets without decrypting them, so a REPEATER can forward traffic on channels whose PSK it does not hold. Role and rebroadcast mode are independent settings; REPEATER is not "formerly ALL_SKIP_DECODING."

Best for: legacy deployments on older firmware where an anonymous, node-list-invisible relay is specifically wanted. For new infrastructure, prefer ROUTER.

ROUTER

The recommended role for stationary, well-placed community infrastructure nodes. A ROUTER is **visible in the node list** and originates its own NodeInfo (and GPS position if configured), which makes it trackable and useful for network monitoring. A ROUTER automatically enables power-saving sleep (this cannot be turned off) and defaults its app connectivity - BLE, Wi-Fi, and Serial - to OFF, so you don't normally text it directly even though it can still originate and receive messages at the protocol level. It is designed for stationary, well-placed nodes.

Best for: fixed community infrastructure and backbone relay nodes where you want visibility on the network and reliable rebroadcasting.

ROUTER_LATE

A relay role that rebroadcasts only after all other nodes have had a chance to do so - it is **not** a prioritized router. It defers to every other node and relays last, which reduces redundant transmissions in dense areas while still ensuring coverage in gaps. Use it for nodes that should rebroadcast only after others.

Best for: filling specific dead spots in a network that already has primary coverage, or as a deliberately low-priority backup relay.

Comparison table

Role	Visible in node list	Rebroadcast strategy	Prioritized routing	Sends own data	Power
CLIENT	Yes	Deferred (managed flooding)	No	Yes (if set)	Moderate

Role	Visible in node list	Rebroadcast strategy	Prioritized routing	Sends own data	Power
REPEATER (deprecated)	No	Once, higher priority, no deferral	Yes	No	High (LoRa active, no force-sleep)
ROUTER	Yes	Once, higher priority, no deferral	Yes	Yes	Force power-saving sleep; app radios off by default
ROUTER_LATE	Yes	After all other nodes	No (relays last)	Yes	High (LoRa always on)

Note: appearing on a public map such as meshmap.net is independent of role - it requires an MQTT uplink to the internet (a gateway), not a particular device role.

Which role should I choose?

- For most ordinary nodes, official Meshtastic guidance is to leave the role at **CLIENT**.
- Use **ROUTER** for fixed, well-placed community infrastructure that should rebroadcast reliably and appear on the network - this is the recommended choice for backbone nodes.
- **REPEATER** is deprecated as of firmware ~2.7.x and is no longer recommended as a default; reserve it for legacy firmware or the narrow case where you specifically want an anonymous, node-list-invisible relay.
- Use **ROUTER_LATE** for a backup node in an area that already has primary coverage, where you want it to relay only after every other node.

Site Planning

Choosing locations, antennas, and using network maps.

Choosing a Repeater Location

Placement determines performance. A well-placed repeater with modest hardware will consistently outperform a poorly placed repeater with expensive equipment.

The primacy of line-of-sight

LoRa signals travel best when there is a clear, unobstructed path between transmitter and receiver. Any obstruction - a building, a ridge, a dense stand of trees - attenuates the signal. The higher your repeater, the more of the surrounding terrain is in line-of-sight.

Location types

Hilltops and ridgelines

The best possible placement. A repeater on a hilltop with 360-degree unobstructed views can serve an area many times larger than the same hardware at ground level. As a rule of thumb, even modest height gains above the surrounding obstructions can noticeably improve coverage; how much depends on the local terrain. The underlying mechanism is the radio horizon, which grows with the square root of antenna height - roughly $3.57 \times \sqrt{h_m}$ km (or about $1.23 \times \sqrt{h_{ft}}$ miles) - so the first few tens of feet above ground level deliver the largest gains.

Rooftops

The most practical option for urban deployments. The highest accessible rooftop in a neighborhood, with the antenna mounted on a short mast, gives excellent urban coverage. Flat commercial rooftops are ideal.

Towers and elevated structures

Communications towers, water towers, and fire lookout towers are excellent platforms. Many communities with amateur radio infrastructure already have tower access - connecting with local ham radio clubs is a good path to shared hosting arrangements.

When co-locating on a registered antenna structure or shared tower, coordinate with the structure owner. RF-exposure (MPE) evaluation and antenna-structure-registration obligations rest with the tower owner, and your Part 15 unlicensed device must not interfere with the licensed services already on the structure.

Mast installations

A 15 - 30 foot mast in a yard or field dramatically improves line-of-sight over the surrounding area. Particularly effective in flat terrain where even modest height above obstructions makes a large difference.

Common placement mistakes

- **Hop gobbling:** A poorly placed repeater that is only marginally better than other nodes can consume hop budget without meaningfully extending range. Every hop used by a marginal relay is a hop unavailable for a more distant leg. Place repeaters where they add significant coverage, not just incremental reach.
- **Too many repeaters too close together:** Dense clusters of repeaters can flood the network with redundant retransmissions. Space repeaters to provide overlapping but not excessively redundant coverage.
- **Ignoring the coverage below:** Very high-gain antennas on tall structures can create dead zones directly beneath them. Size antenna gain to match your deployment height.

Coverage planning tools

- [Meshtastic Site Planner](#) - the official Meshtastic tool; estimates theoretical coverage from a given location
- [HeyWhatsThat](#) - third-party tool for radio horizon visualization based on terrain elevation
- [meshmap.net](#) - third-party community map showing existing Meshtastic nodes near you (only shows nodes reporting to the public MQTT server, so it is incomplete)

Always validate coverage estimates with real-world testing - planning tools do not account for buildings, vegetation, or local RF environment.

Antenna and Signal Range Factors

What determines your repeater's range

Several factors interact to determine how far your repeater can reach. Understanding them helps you make better placement and hardware decisions.

Antenna height and line-of-sight (most important)

This is the dominant factor by a wide margin. Higher placement gives more line-of-sight coverage. Even a few meters of additional height can meaningfully extend coverage. Use terrain analysis tools to identify locations with the best natural line-of-sight before committing to a deployment.

Antenna type and gain

Omnidirectional antennas

Standard for general-purpose repeaters. Higher gain concentrates the signal horizontally, increasing range but reducing coverage of areas directly below. As a common community rule of thumb, many repeater deployments use 3 - 6 dBi omnidirectional antennas as a reasonable balance between reach and overhead coverage; this is not an official Meshtastic specification, so consider your own terrain and coverage goals. Note also that antenna gain counts toward your EIRP limit (see Transmit power below): an antenna over 6 dBi requires a corresponding reduction in

conducted power under FCC rules.

Directional antennas (Yagi)

Best for linking two specific points across a long distance. Directional antennas can achieve dramatically longer range in one direction but provide no coverage off-axis. Useful for point-to-point relay links, not general area coverage.

Antenna and cable quality

Upgrading from a stock antenna to a quality external antenna is often one of the highest-return improvements available. Use short, low-loss coaxial cable (LMR-200 or LMR-400) between the radio and antenna. Long cable runs with cheap coax can negate antenna gain improvements.

LoRa modem presets

Meshtastic provides eight primary preset modem configurations (plus a deprecated ninth, Very Long Slow / `VERY_LONG_SLOW`, which is not recommended and is being phased out). Each preset is a named combination of Spreading Factor (SF), Bandwidth (BW), and Coding Rate (CR) that determines the tradeoff between range, data rate, and airtime. The official preset names are: `SHORT_TURBO`, `SHORT_FAST`, `SHORT_SLOW`, `MEDIUM_FAST`, `MEDIUM_SLOW`, `LONG_FAST`, `LONG_MODERATE`, `LONG_SLOW`, and the deprecated `VERY_LONG_SLOW`.

Preset	SF	BW	CR	Data Rate	Link Budget	Notes
Short Turbo	7	500 kHz	4/5	21.9 kbps	140 dB	500 kHz: legal in the US 902-928 MHz band; restricted in some narrower regional bands
Short Fast	7	250 kHz	4/5	10.9 kbps	143 dB	
Short Slow	8	250 kHz	4/5	6.25 kbps	145.5 dB	
Medium Fast	9	250 kHz	4/5	3.52 kbps	148 dB	
Medium Slow	10	250 kHz	4/5	1.95 kbps	150.5 dB	Recommended for dense networks

Preset	SF	BW	CR	Data Rate	Link Budget	Notes
Long Fast	11	250 kHz	4/5	1.07 kbps	153 dB	Firmware default
Long Moderate	11	125 kHz	4/8	0.34 kbps	156 dB	
Long Slow	12	125 kHz	4/8	0.18 kbps	158.5 dB	Maximum range; very low throughput

Higher link budget = more range. Higher data rate = more network capacity and less airtime per message. Note: these link-budget figures assume a 22 dBm TX power and 0 dBi antennas (the Meshtastic reference conditions); adjust for your actual TX power and antenna gain (e.g. with the Semtech LoRa calculator). They are relative comparisons between presets, not absolute path-loss budgets.

Choosing a preset for your network

The most important rule: match whatever preset the rest of your local network uses. Nodes on different presets cannot hear each other, even on the same channel name. A preset sets SF, bandwidth and coding rate; nodes must match SF and bandwidth to demodulate each other, and because bandwidth also determines the frequency-slot grid, mismatched presets generally also transmit on different center frequencies.

- **Check with your local community first.** Many regional networks have standardized on a specific preset. Check local Discord servers, forums, or network maps before deploying.
- **Long Fast** (firmware default) - widely used, works well for sparse networks and rural deployments. Good starting point if no local standard exists.
- **Medium Slow / Medium Fast** - increasingly common in larger networks (60+ nodes). Faster data rate reduces airtime collisions in dense areas while still covering similar distances.
- **Long Slow** - maximum range, but much lower throughput. Can cause network congestion at scale. Not recommended for regular deployment. (The even slower `VERY_LONG_SLOW` preset is deprecated.)
- **Short Turbo** - highest throughput. It uses 500 kHz bandwidth, which is legal in the US 902-928 MHz band but restricted or prohibited in some narrower regional bands (e.g. EU 868 MHz). US operators may use it; verify compliance for your region first.

Transmit power

More transmit power increases range up to the legal limit. In the US, FCC Part 15 (47 CFR 15.247) caps conducted power at 1 W (30 dBm) in the 902-928 MHz band. Antenna gain up to 6 dBi is allowed at full power. For every dB of antenna gain above 6 dBi, you must reduce conducted power by the same amount (47 CFR 15.247(b)(4)). The frequently quoted 36 dBm (4 W) EIRP figure is simply what 30 dBm + 6 dBi works out to - it is a derived ceiling, not a flat license to radiate 36 dBm with any antenna. The region setting in your firmware is the master control that enforces the legal power cap, so set it correctly. Meshtastic's defaults are compliant with standard antennas, but you remain responsible for EIRP compliance with any non-standard or high-gain antenna - custom setups require the gain-reduction calculation above.

Interference

The 902 - 928 MHz ISM band is a shared, unlicensed band, so other devices operating in it (Wi-Fi extenders, cordless phones, industrial telemetry, and other LoRa networks) can reduce effective range. If you suspect interference, try changing the channel frequency slot within the band and comparing performance.

Using the Meshtastic Network Map

Before deploying a repeater, check a network map to understand where existing coverage exists and where gaps are most significant. This helps you choose a placement that adds the most value to the network. Note that these maps are third-party tools and only show nodes reporting to the public MQTT server, so they are an incomplete, not authoritative, picture of the mesh.

Available map tools

- meshmap.net - community-run map showing nodes that report to the public MQTT server. Third-party and unofficial; coverage is incomplete because it only includes nodes with an internet-connected MQTT gateway.
- **Meshtastic Site Planner** - a coverage-prediction tool for estimating signal reach from candidate locations using terrain data.
- **HeyWhatsThat** - a line-of-sight / viewshed tool useful for checking what a high point can "see" before you commit to a repeater site.

What the maps show

Nodes appear on the map if they have GPS enabled, are configured to share their location, and their data has reached the internet via an MQTT gateway node. Clicking a node shows its ID, name, hardware, and last activity time. Some maps display estimated coverage radius or known links between nodes.

The network map depends on nodes reaching the internet via an MQTT gateway. During internet or grid outages - i.e. during the very emergencies you would most want it - the map can be stale or blank and does **not** reflect live mesh state. Do not rely on it for real-time situational awareness during an incident; verify coverage on the radios themselves rather than trusting the map.

How to use the map for planning

1. Find your area and identify where existing nodes and repeaters are concentrated
2. Identify gaps in coverage - areas with no nearby nodes, or areas that would benefit from a relay between two clusters
3. Look for natural high points near the gap that could serve as a relay location
4. Check whether your planned location already has a node - placing a repeater very close to an existing one adds little value and increases network traffic

Making your repeater appear on the map

A REPEATER-role node suppresses all of its own broadcasts (NodeInfo, position, and telemetry) and is hidden from node lists, so it will not appear on the map. A ROUTER-role node does broadcast its NodeInfo and position by default and will appear on map services that collect MQTT data. If map visibility matters, you can use ROUTER role instead of REPEATER - but choose ROUTER only for genuinely well-positioned infrastructure nodes, and do not switch to ROUTER merely for map visibility, since it also changes power behavior (ROUTER forces power-saving sleep), turns BLE/WiFi/Serial off by default, and affects routing.

Hardware

DIY vs kits, power systems, and enclosures.

DIY vs Pre-built Kits

You can build a Meshtastic repeater from scratch or purchase a pre-built kit. The right choice depends on your budget, technical skills, and time available.

DIY builds

Building from components gives you full control over every aspect of the hardware and can be lower cost if you have relevant skills or existing parts.

Core components needed

- **LoRa board** - ESP32 or nRF52-based board with SX1262 or similar radio (\$20 - \$40). Must support 915 MHz and have an external antenna connector.
- **Antenna** - quality 915 MHz external antenna (\$5 - \$20 for omni, more for high-gain)
- **Weatherproof enclosure** - IP65-rated box with cable glands (\$10 - \$20)
- **Power system** - LiFePO4 battery, solar panel (if remote), charge controller

Note on compliance: the FCC certification of a LoRa module can be tied to specific antenna gains. When you pair a board with your own external antenna, you remain responsible for staying within Part 15 conducted-power limits and the 6 dBi antenna-gain reduction rule (47 CFR 15.247(b)(4)) - for every dB of antenna gain above 6 dBi you must reduce conducted power by the same amount. Use the firmware's region setting to cap power, and reduce conducted power for any antenna over 6 dBi.

Challenges

- Reliable weatherproofing requires attention to detail - water ingress through cable glands or enclosure seals is the most common failure mode
- Power system sizing requires calculation to ensure adequate runtime through cloudy periods
- Soldering may be required depending on the board and power connections

Pre-built kits

Several manufacturers offer kits designed for easy Meshtastic deployment. These trade customization for convenience: weatherproofing is engineered from the factory, power systems are pre-integrated, and setup is primarily software.

Advantages

- Engineered weatherproofing - no DIY enclosure work required
- Integrated power system - battery, solar, and charge controller in one unit
- Faster time to deployment
- Known-good hardware compatibility with Meshtastic firmware

Disadvantages

- Higher upfront cost (\$60 - \$150+)
- Limited hardware customization

Which to choose

Factor	DIY	Kit
Cost	Lower potential, more variable	Higher, more predictable
Setup time	Significant	Minimal
Technical skill needed	Moderate to high	Low
Customization	Full control	Limited
Weatherproofing reliability	Skill-dependent	Generally good to excellent

Power Systems for Repeaters

Repeater and Router role nodes keep the LoRa radio on continuously, which draws significantly more power than a client device that sleeps between uses. Reliable power is a first-class concern for any permanent repeater deployment.

Solar systems

Solar power is the standard for remote deployments without access to mains power. Always-on repeaters draw power continuously and must be sized for worst-case conditions - the shortest winter days, multi-day storms, and cold (which cuts usable battery capacity). Undersized solar is a leading cause of repeaters failing exactly during the prolonged bad-weather emergencies the network is meant to serve. Size for the worst case with several days of autonomy and no solar input, not for average sunny-day draw.

Recommended components

- **Solar panel:** 10 - 30W panel, mounted to maximize year-round sun exposure. South-facing (in the Northern Hemisphere), angled at roughly your latitude for a good annual average. For winter resilience a steeper tilt (about latitude + 15°) is often preferred, because it favors the low sun angles of the short, low-sun months when a repeater is most likely to run short of power.
- **Battery:** LiFePO4 chemistry strongly recommended. It handles temperature extremes, deep discharge cycles, and cycle life far better than LiPo. One critical caveat: LiFePO4 must not be charged below 0°C (32°F) - doing so causes lithium plating and permanent damage or a safety hazard. For cold-climate solar repeaters, use a charge controller/BMS with a low-temperature charge cutoff, or a self-heating battery. A common target is several days (e.g. 3 - 7) of autonomy with no solar input; size it based on your local winter sun-hours and cloud patterns rather than a fixed figure.
- **Charge controller:** MPPT controllers are more efficient than PWM and better suited to variable solar conditions. Sized appropriately for your panel wattage.

Mains power

For rooftop or indoor repeaters with access to building power, a quality regulated 5V or 12V supply is simpler and more reliable than solar. Add a small UPS or battery backup to maintain operation during brief outages.

Software power optimization

Even with always-on radio requirements, you can reduce power draw in software:

- Turn off GPS if the node does not need to report position. The REPEATER role does not broadcast its position, but you may still want to explicitly disable the GPS module to save its power draw.
- Disable the screen/display if present
- Disable Bluetooth: `meshtastic --set bluetooth.enabled false`
- Use the minimum transmit power needed for coverage goals - and never exceed the legal maximum for your region and antenna (see legal considerations: conducted power is capped at 1 W / 30 dBm, reduced dB-for-dB for antenna gain above 6 dBi). Leave the region setting correct, as it is the master legal power cap.
- Choose a balanced modem preset rather than the most aggressive long-range preset (which increases airtime and thus power) - but only among the presets your local network uses. You must run the same modem preset as the rest of your local mesh; nodes on different presets cannot hear each other, so never choose a preset for power reasons alone or you risk isolating your node from the whole local network.

Monitoring battery voltage

For remote deployments, periodically check battery voltage to detect degraded performance before the repeater goes offline. Some Meshtastic nodes can report telemetry data including battery voltage over the mesh.

Configuration

Setting device role, rebroadcast mode, and power optimization.

Setting the Device Role

Configuring a device as a Meshtastic repeater involves two key settings: the device role and the rebroadcast mode. Both can be set using the Meshtastic mobile app or the Python CLI.

Using the mobile app (Android/iOS)

1. Connect to your device in the [Meshtastic app](#)
2. Navigate to **Config** → **Device** → **Role** (app layouts change between versions; this is the current path)
3. Find **Device Role** and select your role. Note that **REPEATER is deprecated as of firmware ~2.7.x**; for most infrastructure nodes the recommended role is now **Router** (which stays visible in the node list), and for the overwhelming majority of ordinary nodes the official guidance is **Client**.
4. Save settings - the device will reboot to apply changes

Using the Python CLI

Install the CLI if you have not already:

```
pip install meshtastic
```

Connect the device via USB, then set the role. The setting key is `device.role` and the value is the bare enum name (for example `ROUTER` or `REPEATER`):

```
# REPEATER is deprecated as of firmware ~2.7.x; ROUTER is recommended for most infrastructure
meshtastic --set device.role ROUTER

# To set the (deprecated) REPEATER role explicitly:
meshtastic --set device.role REPEATER
```

Verify the change:

```
meshtastic --info
```

After setting the role

Once set to REPEATER, the device will no longer appear in other nodes' node lists and does not send NodeInfo (it is anonymous on the mesh) - this is by design. It relays packets without broadcasting its own data, and unlike ROUTER it does not force power-saving sleep. To confirm it is operating, check the serial output or observe that messages are being relayed through it. (If you instead choose ROUTER, the node *does* appear in the node list and the firmware automatically enables power-saving sleep that cannot be turned off, with BLE/WiFi/Serial off by default.)

Flashing Meshtastic firmware

If your device does not have Meshtastic installed, use the [Meshtastic Web Flasher](#) to install firmware directly from Chrome or Edge - no software installation required. Connect via USB, select your device type, and click Flash.

Setting the region

Before the radio will transmit, you must set the correct region. The region setting is the master legal control - it determines both your authorized frequency band *and* the maximum power the firmware will permit:

```
meshtastic --set lora.region US
```

Or in the app: **Config → LoRa → Region → US**. Setting the wrong region (for example leaving a US deployment on EU868) will cause the device to transmit out-of-band on frequencies you are not authorized to use - a clear violation of FCC rules, not just a possibility. In the US, only 902-928 MHz is available for these unlicensed Part 15 devices. Always set the region that matches your physical location before transmitting, and leave `lora.tx_power` at its default of `0` so the firmware applies the region-legal maximum.

Rebroadcast Mode:

ALL_SKIP_DECODING

The rebroadcast mode controls how a node handles incoming packets for retransmission. For repeater deployments, `ALL_SKIP_DECODING` is strongly recommended. Note that `ALL_SKIP_DECODING` is only available when the device role is REPEATER - on any other role (including ROUTER) the setting behaves as `ALL`. If your node uses the ROUTER role, use rebroadcast mode `ALL` instead.

What ALL_SKIP_DECODING does

In this mode, the node rebroadcasts valid LoRa packets without attempting to decode or decrypt them. This mode is only available when `device.role` is REPEATER. It has several important advantages:

- **Lower CPU usage and power draw** - the node skips the packet-decoding/duplicate-check processing entirely, and it never decrypts or re-encrypts payloads
- **Relays channels whose key it does not have** - Meshtastic relays based on the unencrypted packet header, so a repeater forwards packets on any channel that shares its modem settings (preset/frequency), even private channels whose pre-shared key (PSK) - the encryption key that secures a private channel - the repeater does not hold. This lets one public repeater carry traffic for multiple user groups, provided they are all on the same modem preset.
- **Preserves end-to-end encryption** - messages are never decrypted at the repeater; the original encrypted packet is forwarded unchanged, maintaining the security model
- **Faster retransmission** - less processing time per packet means faster relay

Setting ALL_SKIP_DECODING via app

1. Connect to your device in the [Meshtastic app](#)

2. Make sure the device role is set to REPEATER first - the `ALL_SKIP_DECODING` option only applies on the REPEATER role
3. Go to **Settings → Device**
4. Find **Rebroadcast Mode** and select **ALL_SKIP_DECODING**
5. Save and allow the device to reboot

Setting ALL_SKIP_DECODING via CLI

```
meshtastic --set device.rebroadcast_mode ALL_SKIP_DECODING
```

This is only valid when `device.role` is REPEATER. The setting lives under the `device.` namespace, not `lora.`

Other rebroadcast modes (for reference)

Mode	Behavior	Use case
<code>ALL_SKIP_DECODING</code>	Rebroadcasts without decoding or decrypting	Recommended for all repeater deployments; requires REPEATER role
<code>ALL</code>	Processes/validates packets (including duplicate detection) before rebroadcasting the original encrypted packet; does not decrypt or re-encrypt payloads	Default; used on ROUTER and other roles that should relay
<code>LOCAL_ONLY</code>	Does not relay to other nodes	Not suitable for repeaters
<code>NONE</code>	No rebroadcasting (only permitted for SENSOR/TRACKER/TAK_TRACKER roles)	Not suitable for repeaters

Power Optimization Settings

Repeater and Router role nodes keep the LoRa radio on continuously, which draws significantly more power than a client device that sleeps between uses. These settings minimize everything else to extend runtime on battery or solar power.

Disable Bluetooth

Once configured, a dedicated repeater does not need Bluetooth. Disabling it saves power and removes an unnecessary wireless interface.

```
meshtastic --set bluetooth.enabled false
```

Disable GPS (REPEATER role handles this automatically)

The REPEATER role does not broadcast position data, but the GPS module may still draw power unless you disable it explicitly in the Position config. If your board has a GPS module, disable it to save power:

```
meshtastic --set position.gps_mode DISABLED
```

Disable the screen

If your device has a display, minimize the screen-on time. Display backlights draw meaningful power. Note that `display.screen_on_secs 0` does **not** turn the screen off - 0 means the 10-minute default. To minimize screen-on time, set a small non-zero value instead:

```
meshtastic --set display.screen_on_secs 1
```

Set appropriate transmit power

Transmit power is the largest variable power draw. In most cases, leave `tx_power` at its default of **0**, which tells the firmware to use the maximum power your region legally permits for this hardware - the region setting is what enforces the legal cap. Only set an explicit value if you are deliberately reducing power (for example, to limit interference with nearby nodes). Higher power is not always better.

```
meshtastic --set lora.tx_power 0
```

If you do set an explicit value, `tx_power` is conducted PA power; stay at or below 30 dBm. If you use an antenna over 6 dBi gain, you **MUST** reduce conducted power by 1 dB for every dB of gain above 6 dBi (47 CFR 15.247(b)(4)). Actual radiated power also depends on your board's PA capability and feedline loss, so tie any fixed value to your antenna-gain / EIRP calculation rather than picking a number arbitrarily.

Use the right modem preset - but match your local network

Modem preset affects how long the radio is transmitting each packet, which directly impacts power consumption:

- Slower, longer-range presets (Long Slow, Long Moderate) keep the radio transmitting longer per packet - higher average power draw
- Faster presets (Medium Slow, Medium Fast) transmit each packet more quickly - lower average power draw, and better network capacity in dense areas
- Long Fast (firmware default) sits in the middle

Do not choose a preset based on power alone - you must use the same preset as the rest of your local network. Check what preset your local community or regional network uses before deploying. See the [Choosing a Modem Preset](#) page for guidance.

Managed Mode (optional)

For a deployed repeater that should not be reconfigured by whoever is near it, you can enable Managed Mode (`security.is_managed`). This blocks client apps from writing config; changes must then come via PKC Remote Admin (firmware 2.5+) or the legacy Admin channel. **Verify that**

remote admin works before enabling this, or you risk locking yourself out - once managed mode is on, you cannot reconfigure the node from a normal client app.

```
meshtastic --set security.is_managed true
```

Choosing a Modem Preset

Your modem preset is one of the most consequential configuration decisions for your repeater. It must match every other node on your channel - and different regional communities have standardized on different presets.

What a preset controls

Each preset is a named combination of three LoRa parameters:

- **Spreading Factor (SF):** Higher SF = longer range, longer airtime per packet. Range from 7 (fastest) to 12 (longest range).
- **Bandwidth (BW):** Wider bandwidth = faster data rate, slightly less range.
- **Coding Rate (CR):** Higher CR adds forward error correction overhead.

The full preset table

Preset	SF	BW	CR	Data Rate	Link Budget
Short Turbo	7	500 kHz	4/5	21.9 kbps	140 dB
Short Fast	7	250 kHz	4/5	10.9 kbps	143 dB
Short Slow	8	250 kHz	4/5	6.25 kbps	145.5 dB
Medium Fast	9	250 kHz	4/5	3.52 kbps	148 dB
Medium Slow	10	250 kHz	4/5	1.95 kbps	150.5 dB
Long Fast	11	250 kHz	4/5	1.07 kbps	153 dB
Long Moderate	11	125 kHz	4/8	0.34 kbps	156 dB
Long Slow	12	125 kHz	4/8	0.18 kbps	158.5 dB

The link-budget figures above assume 22 dBm TX power and 0 dBi antennas; your real budget shifts with your actual transmit power and antenna gain. `VERY_LONG_SLOW` (SF12 at the narrowest bandwidth) is the only preset offering even more range than Long Slow, but it is deprecated and not recommended by the Meshtastic project for regular use.

Link budget is roughly the maximum single-hop path loss the signal can survive, computed at 22 dBm TX and 0 dBi antennas; real budgets shift with your actual TX power and antenna gain. Higher is more range. Data rate is the raw channel throughput - faster data rate means each transmission occupies less airtime, reducing collisions in busy networks.

Step 1: Ask your local community

Before choosing a preset, check what your local or regional network has standardized on. **All nodes on a channel must use the same preset to communicate.** If your area already has an established mesh, deploying on a different preset will isolate your repeater from it entirely.

Check local Discord servers, third-party tools such as the community [network map at meshmap.net](https://meshmap.net) (unofficial; only shows nodes reporting to the public MQTT server), or regional mesh community pages for your area's standard.

Step 2: Match your network density

If no local standard exists, choose based on your expected network density:

Sparse / rural networks (under ~60 nodes in range)

Long Fast (the firmware default) is a reasonable choice. Range is maximized and the lower data rate is not a problem when relatively few nodes are generating traffic.

Larger networks (more than ~60 nodes)

Consider Medium Slow or Medium Fast. The official guidance is to consider moving away from Long Fast once a mesh exceeds roughly 60 nodes. The 3 - 4x higher data rate significantly reduces collision probability while still maintaining respectable, only slightly reduced range compared to Long Fast.

Dense urban networks (well over 60 nodes)

Faster presets are strongly preferred. Long Fast in a 100+ node network can cause congestion as packet airtime accumulates. Medium Slow has been reported as the standard for at least one large community mesh: the Meshtastic Bay Area community (reportedly 150+ nodes, as of 2025) is reported to have migrated to Medium Slow with improved reliability, and the Wellington Region Mesh (New Zealand) is reported to use Short Fast. These community figures are unsourced and change over time - check your local community's current standard rather than relying on these examples.

Presets to avoid for repeater deployment

- **Long Slow / Very Long Slow** - maximum range but extremely long airtime per packet. Can saturate a network even at low message rates. Long Slow and the even slower, deprecated `VERY_LONG_SLOW` are not recommended for regular use by the Meshtastic project.
- **Short Turbo** - highest throughput, but it uses 500 kHz bandwidth. This is legal in the US 902-928 MHz band, but some narrower regional bands (for example EU 868 MHz) do not permit it. Verify legality for your region before use.

Setting the preset via CLI

```
meshtastic --set lora.modem_preset MEDIUM_SLOW
```

Valid preset names: `SHORT_TURBO`, `SHORT_FAST`, `SHORT_SLOW`, `MEDIUM_FAST`, `MEDIUM_SLOW`, `LONG_FAST`, `LONG_MODERATE`, `LONG_SLOW`, `VERY_LONG_SLOW` (`VERY_LONG_SLOW` is deprecated / not recommended).

Frequency slot and channel name

Meshtastic derives the transmit frequency from the channel name hash by default (frequency slot 0). This means two nodes with the same channel name and same preset will automatically land on the same frequency. You can override this with a specific slot number if needed, but the default hash-based behavior is correct for most deployments.

Best Practices

Avoiding common problems, legal considerations, and config backups.

Deployment Best Practices

Plan before you deploy

A poorly placed repeater can add network congestion without meaningfully extending coverage. Before deploying:

- Check the network map for existing nodes near your intended location
- Use coverage- and line-of-sight-planning tools (such as the Meshtastic Site Planner and HeyWhatsThat) to estimate coverage from candidate locations - see the network-map page for the full list of planning tools
- Walk or drive the coverage area with a personal node to measure real-world signal before committing to a permanent install

Avoid hop gobbling

Each relay uses one hop ("hop gobbling" means a relay that uses up a hop without adding meaningful range). If a repeater is only marginally better positioned than surrounding nodes, it will consume a hop without significantly extending message range. Place repeaters where they add substantial coverage - bridging a gap or reaching a new area - not just adding a small increment to existing coverage.

Spacing between repeaters

Overlapping coverage is good for resilience, but too many repeaters within range of each other creates redundant retransmissions that congest the network. A general guideline is to space repeaters so each covers a distinct area, with just enough overlap for redundancy.

Test after deployment

After installing, walk or drive the intended coverage area and verify that messages successfully reach destinations through the repeater. Test at the edges of expected coverage, not just close in. Note that a quiet walk-test proves coverage, not capacity: for networks intended for emergencies, also validate behavior under realistic concurrent traffic (multiple users sending at once), and re-test coverage periodically, since foliage, new construction, and changing neighbor RF can all shift it over time.

Document your deployment

Record the hardware, firmware version, configuration, location coordinates, and power system details for each repeater you deploy. This makes firmware updates, troubleshooting, and handoff to other maintainers much easier. Consider contributing your repeater location to the community map.

Maintain firmware

Meshtastic releases updates regularly with performance improvements, bug fixes, and new features. Keep your repeater firmware updated using the web flasher at flasher.meshtastic.org.

Legal Considerations

Meshtastic and MeshCore both operate in license-free ISM radio bands, but license-free does not mean unregulated. You must comply with applicable FCC rules for US deployments.

US regulatory framework (FCC Part 15)

In the United States, 915 MHz operation is governed by FCC Part 15 rules for ISM band devices. The key limits for intentional radiators in the 902 - 928 MHz band:

Parameter	Limit
Conducted transmit power	1 W (30 dBm) maximum, referenced to an antenna of 6 dBi or less
EIRP (with antenna gain)	36 dBm (4 W) is a <i>derived</i> ceiling - it is the result of 30 dBm conducted + 6 dBi gain, not a flat standalone cap. For antenna gain above 6 dBi, conducted power must be reduced dB-for-dB (see below).
Duty cycle	US Part 15 imposes no duty-cycle limit on digital-modulation systems such as LoRa in 902 - 928 MHz. The dwell-time limits in 15.247(a)(1) apply only to frequency-hopping (FHSS) systems, not to LoRa's digital modulation. Other regions (e.g. the EU 868 MHz band) DO impose duty-cycle limits - check your region.

Antenna gain and EIRP

EIRP combines transmit power and antenna gain: $EIRP = \text{conducted power} + \text{antenna gain (in dBi)} - \text{cable loss}$.

The actual FCC rule (47 CFR 15.247(b)(4)) is: conducted transmit power of **1 W (30 dBm) is the absolute maximum**, and antenna gain up to 6 dBi is permitted at that full power. **For every dB of antenna gain above 6 dBi, conducted power must be reduced by the same number of dB.** The often-cited 36 dBm EIRP figure is simply the result of this rule at exactly 6 dBi of gain (30 dBm + 6 dBi = 36 dBm) - it is not an independent allowance to radiate 36 dBm EIRP with any

antenna.

Worked examples:

- 3 dBi antenna: the 30 dBm conducted limit is absolute - you may **not** raise conducted power above 30 dBm to reach 36 dBm EIRP.
- 9 dBi antenna (3 dB over 6 dBi): reduce conducted power by 3 dB to 27 dBm (500 mW). Result: 36 dBm EIRP - at the legal limit, and compliant because conducted power was reduced per the 6 dBi rule (not merely because EIRP lands at 36 dBm).
- 9 dBi antenna + 30 dBm (1 W) conducted = 39 dBm EIRP and also violates the dB-for-dB conducted-reduction rule - **over the limit**.

Meshtastic ships with a region-based power cap; setting `tx_power` to 0 lets the firmware use the maximum the selected region allows. This is compliant only when paired with an antenna of 6 dBi or less. With any higher-gain antenna you must reduce conducted power per the 6 dBi rule above. The region setting is the master legal power-cap control - set it correctly first.

No amateur radio license required

Standard Meshtastic and MeshCore operation in [the 915 MHz ISM band](#) does not require an amateur radio license. The band is available to any compliant Part 15 device without licensing. This holds only while the device remains within its Part 15 certification: adding a non-certified antenna or amplifier, or exceeding the power/EIRP limits, can void the certification - at which point the operation is no longer authorized under Part 15. The operator is responsible for keeping the device compliant, and Part 15 devices must accept interference and must not cause harmful interference.

Always verify current regulations

Radio regulations can change. The information above is provided as a general guide. For definitive requirements, consult the [FCC Part 15 rules](#) directly.

Backing Up Your Configuration

Before making changes to a working repeater - or before a firmware update - back up the configuration. This saves your channel keys, LoRa settings, and device role, allowing full recovery if something goes wrong.

Export configuration via CLI

```
pip install meshtastic # if not already installed
meshtastic --export-config > my_repeater_config.yaml
```

The `--export-config` command produces a YAML file (not JSON). This is the canonical, restorable backup format - use it consistently. The exported YAML file contains:

- Node ID and long/short name
- LoRa settings (region, modem preset, rebroadcast mode, TX power)
- Channel names and pre-shared keys (PSKs)
- Device role
- Power and display settings

Restore configuration

```
meshtastic --configure my_repeater_config.yaml
```

Restore with `--configure`, which is the counterpart to `--export-config`. There is no `--import-config` flag in the Meshtastic CLI. Note also that `meshtastic --info` output is only a human-readable status dump and is **not** a restorable backup - always use `--export-config` for backups.

Store backups safely

The configuration file contains your channel PSKs. Store it in a secure location. Do not share it publicly or commit it to a public repository - anyone with the PSK for a private channel can read its messages.

After a firmware update

Firmware updates sometimes reset device settings. Always verify device role, rebroadcast mode, and LoRa region after updating. If settings are lost, restore from your backup YAML file with

```
meshtastic --configure my_repeater_config.yaml.
```

Network Troubleshooting

Diagnosing Meshtastic Network Problems

This guide covers systematic diagnosis of common Meshtastic network issues: nodes that can't hear each other, poor range, network congestion, and routing failures.

Diagnostic framework

Work from the bottom up: radio layer first, then routing, then application.

1. **Radio layer:** Can the nodes physically hear each other? Check RSSI/SNR.
2. **Configuration:** Are both nodes on the same preset and channel?
3. **Routing:** Is the path between nodes working? Use Trace Route.
4. **Application:** Is the app connected properly? Is the message actually sending?

Problem: Two nodes can't communicate

Check 1: Same preset?

Nodes on different presets cannot hear each other. Verify the preset directly on each node:

```
meshtastic --get lora.modem_preset # run this on each node
```

Both nodes must report the same preset (e.g., LONG_FAST or MEDIUM_SLOW). You generally cannot read a remote node's modem preset from a node-list dump - check each node on its own connection. If they differ: change one to match the other.

Check 2: Same channel and PSK?

Nodes must be on the same channel with the same PSK. The default public channel (LongFast) uses the well-known default PSK `AQ==` (a publicly known, weak key) - it is *not* an empty/no-encryption key. If you've customized your channel, the other node needs the same configuration.

```
meshtastic --info # Shows channel list with PSK hashes
```

PSK hash mismatch = nodes won't communicate even if physically in range.

Check 3: Physical range and line of sight

In the [Meshtastic app](#), check if the node appears in the node list with any RSSI/SNR value. Do not assume a low or negative SNR means a bad link: LoRa decodes well below the noise floor, with per-spreading-factor demodulation limits running from about -7.5 dB (SF7) to about -20 dB (SF12). At Long Fast / Long Slow, an SNR of -10 to -18 dB is routinely decodable, not "at the noise floor." A link only becomes marginal as SNR approaches the per-SF demod limit (for example below about -17 dB at SF11/Long Fast), or as RSSI approaches the receiver sensitivity floor (the SX1262 is sensitive to roughly -137 dBm at SF12). If the node doesn't appear at all, no packets are being received.

Test: bring both devices to within 100 feet of each other (no obstacles). If they communicate at that distance, it's a range/obstruction problem. If they still can't communicate at 100 feet, it's a configuration problem.

Problem: Intermittent message delivery

Check: Hop count

Messages requiring more hops have higher failure rates. Check your hop limit setting:

```
meshtastic --get lora.hop_limit
```

Default is 3 (the maximum is 7). Before raising it, verify with a traceroute that the path actually needs more hops. Raising `hop_limit` increases airtime and channel utilisation and can worsen congestion rather than fix dropped messages — the preferred fix for a long path is usually to add infrastructure to shorten it. If a traceroute confirms the path genuinely needs more hops, raise it incrementally (for example to 4) and monitor channel utilisation:

```
meshtastic --set lora.hop_limit 4
```

See [Hop Limit Configuration for Repeaters](#) for the full caveats on high hop counts and channel utilisation.

Check: Network congestion

In dense networks (20+ nodes), Long Fast preset can cause congestion. Symptoms: high message drop rate even with good RSSI, large delays. Solution: migrate to Medium Slow or Medium Fast preset. Requires coordination with all network participants - all nodes must change at the same time.

Check: Router node availability

If a critical router node goes offline, messages that depended on that path fail. This is a design red flag, not just a diagnostic finding: if a single router node failing kills delivery for part of your network, that node is a single point of failure — see [Designing for Reliability \(N+1 Redundancy\)](#). For emergency use, no incident-critical route should depend on one node. Use Trace Route before and after to confirm the path change:

```
meshtastic --traceroute !nodeId
```

Problem: Poor range from a new repeater

1. **Antenna connected?** An unconnected antenna transmits into the PCB and can damage the radio front-end. Verify the antenna is finger-tight. Some boards have a separate BLE antenna and LoRa antenna - ensure both are connected.
2. **Correct antenna type?** Most boards use SMA or u.FL connectors. Verify your antenna has the correct connector type and is rated for your operating band - use a 915 MHz (902-928 MHz) antenna in North America. An 868 MHz antenna will radiate at 915 MHz but with a shifted resonance and elevated VSWR; for a broadband whip the penalty is usually small (a fraction of a dB), but for a narrowband tuned antenna the loss and reflected power can be significant, and high VSWR stresses the power amplifier. Note that antenna gain also bears on legal power limits — antennas over 6 dBi require a dB-for-dB reduction in conducted power per FCC 47 CFR 15.247(b)(4).
3. **TX power set correctly?** Check that TX power hasn't been set to an unusually low value:

```
meshtastic --get lora.tx_power
```

4. **Obstructions:** Even a metal enclosure, HVAC equipment, or tree canopy directly around the antenna can reduce range significantly. Test with the antenna in the clear before committing to a location.

Repeater Performance and Maintenance

A deployed repeater requires periodic attention to maintain performance. This page covers the key maintenance tasks and performance metrics for Meshtastic infrastructure nodes. Throughout, "infrastructure node" means the general concept; **ROUTER** and **REPEATER** refer to the specific device roles (which behave differently - a ROUTER appears in the node list, a REPEATER does not).

Key performance indicators

The targets below are **recommended operational goals set by the author**, not official Meshtastic or manufacturer specifications. Treat them as starting points, not hard specs.

Metric	Healthy range	Action if outside range
Node uptime	>95% over 30 days (baseline only)	>95% is a baseline health indicator, not a reliability guarantee - it still allows ~36 hours/month offline, and says nothing about <i>when</i> the downtime falls. For nodes an emergency operator depends on, investigate ANY unexplained downtime and treat repeated outages as disqualifying for sole-path infrastructure. Redundancy, not a high uptime number, is what guarantees coverage during an incident.
Average RSSI to neighbors	-70 to -100 dBm typical	RSSI down to roughly -120 to -130 dBm can still be decodable depending on spreading factor (SX1262 sensitivity ~-137 dBm at SF12). Judge link health by SNR margin and packet success, not RSSI alone; a sudden RSSI drop versus a known baseline is the useful warning sign, not a fixed -110 dBm threshold.

Metric	Healthy range	Action if outside range
SNR to nearest neighbor	SNR margin above the per-SF demod limit	LoRa decodes well below 0 dB SNR - demod limits run from ~ -7.5 dB (SF7) to ~ -20 dB (SF12). What matters is SNR margin above the per-SF limit. For Long Fast (SF11, limit ~ -17.5 dB), SNR down to about -12 dB is comfortable and approaching -17 dB is marginal. A backbone link at -10 to -12 dB SNR is normal and healthy, not a fault - do not treat $\text{SNR} < 0$ dB as "noise floor / unreliable."
Battery voltage (solar)	Depends on pack configuration	State your battery configuration before reading these numbers. A <i>single</i> LiFePO4 cell has a working range of roughly 3.2 - 3.6 V; a 4S LiFePO4 pack is ~ 12.8 V nominal; a single Li-ion runs 3.0 - 4.2 V. Use the per-chemistry, per-configuration cutoff for your pack and align it with the cutoffs on the remote-monitoring page (e.g. ~ 11.5 V for a 12 V lead-acid pack, ~ 3.0 V per cell for LiPo). Repeated readings near the low end of <i>your</i> pack's range indicate an undersized power system.
Packets forwarded per hour	Varies by location	Sudden drop to 0 = node possibly offline

Routine maintenance checklist (quarterly)

- Check the node appears on a current community/third-party map (such as the official Meshtastic map or a community MQTT-based map) - verify the service is still live, as these third-party maps come and go. Note that a node set to the REPEATER role will **not** appear on node maps (it is hidden from the nodes list); a ROUTER will. These maps only show nodes reporting to a public MQTT server and are not authoritative.
- Verify RSSI/SNR to neighboring nodes hasn't significantly degraded versus your baseline
- Check battery voltage logs if monitoring - look for downward trend
- Inspect solar panel: clean off debris, verify no shading from new growth
- Check antenna connector for corrosion or loosening (especially after winter)
- Verify firmware version - update if significantly behind current release
- Check enclosure for water intrusion - condensation inside is an early warning sign

Firmware update process for deployed nodes

Flashing new firmware (reflashing the binary) on a deployed repeater requires physical USB/serial access. Configuration changes can be made remotely via Remote Admin over the mesh, but a firmware reflash cannot be done remotely. Prepare:

1. **Before taking ANY repeater offline for a firmware update, confirm an alternate path covers its users** (run traceroutes from both sides). NEVER update the only repeater serving an area without a tested spare on hand or a deployed temporary relay - a failed flash can leave the firmware in a crash loop, knocking out coverage with no rollback. Bring a known-good spare and a copy of the prior firmware so you can roll back on-site if the flash fails.
2. Schedule a maintenance window and notify the community (the node will be offline during update)
3. Bring: laptop, USB cable for your device type, and the firmware binary or web browser access
4. Before disconnecting: record current configuration (region, TX power, role, channel settings, position) with a restorable backup: `meshtastic --export-config > config_backup.yaml`. (Do **not** rely on `meshtastic --info` - that is only a human-readable status dump and cannot be re-imported.)
5. Flash new firmware via web flasher (flasher.meshtastic.org)
6. Verify settings after flash - firmware updates occasionally reset some settings to defaults
7. Confirm node reappears on the network before leaving the site

Common hardware failures

Symptom	Likely cause	Fix
Node gone offline after storm	Water intrusion, lightning strike, blown fuse	Inspect enclosure, check fuse, examine for burn marks on PCB
Range suddenly reduced	Antenna connector loosened or corroded	Re-seat antenna, check connector for oxidation, replace if needed
Frequent reboots	Power supply instability (low battery/solar)	Check battery voltage, check charge controller output
Firmware crash loop	Corrupted flash or incompatible firmware	Factory reset and reflash
BLE not discoverable	BLE antenna loose (V3 only); software issue	For V3: reseal u.FL BLE antenna. Otherwise reflash.

When to replace vs. repair

LoRa boards are inexpensive (\$15 - 75). General guidance:

- Physical damage to SMA connector or RF front-end: replace board. Repair costs often exceed replacement.
- Software issue (firmware bugs, configuration corruption): reflash before considering hardware replacement.
- Battery degradation (LiFePO₄): replace battery after 5+ years or when capacity drops below 70% of original.
- Solar panel degradation: typical panels lose 0.5% efficiency per year. Replace if output is more than 20% below original spec after 10+ years.

Meshtastic Repeater Setup

Step-by-step setup, role selection, and channel configuration for Meshtastic repeaters.

Router vs. Repeater Role — Which to Choose

Overview of Device Roles

Meshtastic firmware supports several device roles that control how a node behaves on the mesh. For infrastructure nodes the historically relevant roles are **ROUTER** and **REPEATER**, but note that **REPEATER is deprecated as of firmware ~2.7.x**. For most stationary, well-placed infrastructure the current recommendation is **ROUTER** (or **ROUTER_LATE** for a node that should rebroadcast only after others have had a chance); for the overwhelming majority of ordinary nodes the official guidance is simply **CLIENT**.

ROUTER Role

- Always rebroadcasts each packet once (does not defer to a neighbor that rebroadcasts first), with prioritized routing. The scope of what it rebroadcasts is governed by `rebroadcast_mode` (default ALL), not by the role itself.
- Appears in the app node list of nodes that hear it. It will only appear on meshmap.net if its packets reach the internet through an MQTT gateway somewhere on the mesh - merely being a ROUTER does not put it on the public map.
- At the protocol level a ROUTER can still originate and receive messages, but BLE/WiFi/Serial app connectivity is OFF by default, so you do not normally text it directly to check status without first enabling an interface.
- Uses more airtime than deferring roles because it always rebroadcasts (it never skips when a neighbor already did), not because it sends extra copies of a packet.
- Rebroadcasts with priority by using a shorter contention delay ("cutting in line") so it relays before other nodes, which helps extend range. It does not prioritize by remaining hop count.
- Automatically enables power-saving sleep on ESP32 (this cannot be turned off) and is designed for stationary, well-placed nodes.
- **Good for:** permanent, strategically placed stationary nodes acting as backbone relays; medium-traffic networks where a preferred relay improves reach.

REPEATER Role

REPEATER is deprecated as of firmware ~2.7.x. Prefer ROUTER (or ROUTER_LATE) for new infrastructure. The behavior below is retained for reference and for existing deployments.

- Like ROUTER, it is a preferred relay that always rebroadcasts each packet once with a shorter delay; it does not prioritize by remaining hop count and is not a "hop maximiser."
- Does **not** broadcast its own position or send NodeInfo - it is anonymous on the mesh and does not appear in the node list, which reduces network traffic.
- Is not normally addressed for "self" - you do not text it directly. This is a default-configuration behavior, not the firmware omitting a send code path.
- Does not force power-saving sleep, so it keeps its LoRa radio on. Baseline power draw is therefore similar to ROUTER; differences between the two roles are marginal (self-generated airtime, CPU/RAM) rather than one drawing dramatically less than the other.
- **Good for (legacy):** permanent unattended relay-only infrastructure where the node should stay anonymous on the mesh. For new builds, use ROUTER instead.

When to Use Each Role

Scenario	Recommended Role
Temporary deployment or field relay that doubles as a usable client	CLIENT
Node needs to appear in the node list for coordination	ROUTER
Permanent mountaintop or rooftop installation	ROUTER
Solar-powered, unattended backbone node	ROUTER
Ordinary user node	CLIENT

CLIENT_MUTE Role

CLIENT_MUTE prevents a regular client device (phone, laptop) from rebroadcasting mesh traffic - a normal CLIENT still rebroadcasts under managed flooding, but CLIENT_MUTE does not relay at all. This is useful for devices that are on the mesh but should not consume airtime acting as relays, such as tablets used only for monitoring.

Setting the Role

In the [Meshtastic app](#): on Android, **Settings → Device → Role**; on iOS/macOS, **Settings → Device Configuration → Device → Role**. The exact menu path can vary by app version.

Via the CLI: `meshtastic --set device.role ROUTER` (the key is `device.role` and the value is the bare enum name, e.g. `ROUTER`, `ROUTER_LATE`, `CLIENT`).

Via the web client at client.meshtastic.org: connect via USB, navigate to Device Config, and choose the desired role from the drop-down.

Initial Setup Walkthrough

Prerequisites

- A Meshtastic-compatible device (T-Beam, Heltec, RAK WisBlock, etc.) flashed with current firmware - use the official flasher at flasher.meshtastic.org.
- A phone with the [Meshtastic app](#) (iOS or Android) **or** a Chrome-based browser for the web client.
- An antenna connected *before* powering the device.

Step 0 - Set the Region (do this first)

Config → LoRa → Region - set this to your country (e.g.) before transmitting. The region is the master legal control: it determines both your authorized frequency band *and* the maximum power the firmware will permit. The device should not be used to transmit until the region is set correctly - in the US, only 902-928 MHz is available for these unlicensed Part 15 devices, and an incorrect region will transmit out-of-band on frequencies you are not authorized to use. Leave at its default of so the firmware applies the region-legal maximum.

Step 1 - Connect to the Device

Via Bluetooth: Open the Meshtastic app, tap the + button, and scan for your device. Pair when prompted.

Via USB: Navigate to client.meshtastic.org in Chrome, click **New connection → Serial**, and select the device's COM/serial port.

Step 2 - Set the Device Name

Navigate to **Config → Device**:

- **Long Name:** Your full identifier for this node, e.g. `KD9XYZ-Hilltop-1`. This appears in message headers and node lists.
- **Short Name:** Up to 4 bytes (typically 4 ASCII characters), shown on the map, e.g. `H1`. Note that multi-byte UTF-8 characters such as emoji or non-ASCII text consume more than one byte each, so fewer than 4 will fit.

Step 3 - Set the Device Role

Config → Device → Role. Use **ROUTER** when you want the infrastructure node to stay visible in the Nodes list and topology (the recommended choice for most infrastructure); use **REPEATER** for a pure relay that is invisible in the Nodes list and originates no NodeInfo, Position, or Telemetry. Note that **REPEATER is deprecated as of firmware ~2.7.x**, so ROUTER is preferred for new deployments; for ordinary non-infrastructure nodes the official guidance is CLIENT. Both ROUTER and REPEATER are infrastructure relay roles - neither is specifically a "monitoring point." See the *Router vs. Repeater Role* page for guidance.

Step 4 - Configure the Channel

The default channel (*LongFast* or *Default*) works for joining the public mesh. To match a local community's private channel:

1. Navigate to **Channels → Channel 0**.
2. Set the **Name** and **PSK** to match the local standard. The **PSK** (pre-shared key) is the channel's encryption key - all nodes that share a channel must use the same key to read each other's messages. Your local community provides this key, usually as a shared QR code or channel URL that imports the name and key together; you rarely need to type a raw key by hand. A PSK can be 0 bytes (no encryption), 16 bytes (AES-128), or 32 bytes (AES-256). See the channel-configuration page for details.
3. Contact your local Meshtastic community (Discord, or a local club) for the channel name and key.

Step 5 - Set the Modem Preset

Radio Config → LoRa → Modem Preset - select the preset your local network uses (typically *Long Fast* or *Medium Slow*). **Critical: all nodes on the same network must use the same modem preset or they cannot hear each other.**

Step 6 - Configure a Fixed Position

For an unattended repeater without GPS, set a fixed position so the node can report its location:

1. **Config → Position → Fixed Position → Enable.**
2. Enter the latitude, longitude, and altitude of the deployment site (look up coordinates with any map app).
3. Set **Position Broadcast Interval** to a long value for a static node - `43200` seconds (12 hours) or `86400` seconds (24 hours) is appropriate, since a fixed node's position never changes and frequent rebroadcasts waste airtime.

Step 7 - Power Optimisations (Battery / Solar)

- **Config → Power** - set sleep and minimum wake intervals to the lowest practical values.
- Automatic forced power-saving is tied to the **ROUTER** role (ESP32 only), not REPEATER - the firmware enables it for ROUTER and it cannot be turned off. The REPEATER role does not force power-saving sleep; tune power settings manually where the platform supports it.
- **Screen Timeout** (`display.screen_on_secs`) controls how long the display stays on after activity. Lowering it reduces draw, but note that a value of `0` selects the firmware default (about 10 minutes), *not* "always off" - set a small non-zero value if you want the screen to turn off quickly.

Step 8 - Disable Bluetooth (Optional)

Config → Bluetooth → Enabled → false. This saves power and removes an attack surface on unattended nodes. **Note:** once Bluetooth is disabled, you will need a USB/serial connection or remote admin (PKC admin keys, firmware ≥ 2.5) to re-enable it.

Step 9 - Verify Operation

- Watch the node list for other nearby nodes appearing - this confirms receive is working.

- Send a test message and verify it is received on another device.
- If the node is uplinked to MQTT through an internet gateway, it may also appear on the third-party community map meshmap.net. Note that meshmap.net is not an official Meshtastic property and only shows nodes whose data reaches it via an MQTT gateway - a node with no internet gateway may never appear there, and appearance timing is not guaranteed.

Channel Configuration for Infrastructure Nodes

What Are Channels?

Meshtastic supports up to **8 simultaneous channels** (numbered 0 - 7). Channel 0 is the *primary channel* used for most mesh traffic. Channels 1 - 7 can carry separately encrypted traffic for specific groups or purposes.

PSK - Pre-Shared Key

Each channel has a **name** and a **pre-shared key (PSK)**. The PSK can be **0 bytes** (no encryption), **16 bytes** (AES-128), or **32 bytes** (AES-256) - it is not always a 32-byte AES-256 key. Two nodes can communicate on a channel only if they share the same name *and* the same PSK (same key length and value). The PSK encrypts the channel payloads with AES (AES-128 for a 16-byte key, AES-256 for a 32-byte key).

The Default Public Key

The *Default* public channel (LongFast) uses a **well-known, publicly published default key** (the 1-byte key `AQ==`, 0x01) - it is a defined weak key, not an empty/no-encryption PSK, so traffic on it is effectively public. Any node running Meshtastic with the default channel can participate in the public mesh. Using a **custom PSK** creates a private channel readable only by nodes that hold that key.

Channel Strategy for Community Infrastructure Nodes

- **Channel 0 - Default PSK:** Keep the primary channel on the public Default key so all community users benefit from your repeater's coverage.
- **Channel 1 - Private PSK:** Adding a secondary channel with a private key for your personal use or club coordination is acceptable. The repeater will relay packets on both channels.

Remote Administration

Meshtastic supports remote administration so that nodes can send configuration commands - changing settings without physical access to the device. In firmware 2.5 and later, the recommended method is **PKC admin keys** configured under **Security Config**: you add the public key(s) of trusted administrator nodes to the remote node's admin-key list, and those nodes can then administer it. The legacy **admin channel** (a secondary channel named exactly `admin`, case-sensitive) exists only for managing pre-2.5 nodes; there is *no* "Is Admin" toggle on a channel.

Setting up remote administration is strongly recommended for unattended permanent deployments.

1. On firmware 2.5+, open **Security Config** and add the public key of each trusted administrator node to the node's admin-key list. (For legacy pre-2.5 nodes only: create a secondary channel named exactly `admin`.)
2. Verify the exact app path before you publish or commit the change - getting remote admin wrong can lock you out of a remote node entirely.
3. Back up the configuration (export the config, and for legacy admin channels save the QR code) and store it securely - anyone with the admin keys or legacy admin-channel QR code can reconfigure your node remotely.

Channel Propagation

What a relay rebroadcasts depends on its **Rebroadcast Mode**, not just on which channels it has configured. Meshtastic relays based on the *unencrypted* packet header, so under the default `ALL` mode a repeater rebroadcasts **all** packets that match its modem settings and frequency - *including packets on channels whose PSK it does not have and cannot decrypt*. This means a private channel can be carried by a public repeater. Only the `LOCAL_ONLY` or `KNOWN_ONLY` rebroadcast modes restrict relaying to the repeater's own configured/known channels. (A node still has to be on the same modem preset and frequency to hear and relay at all.) For a public community repeater, keeping only the public Default channel configured is the standard approach, but be aware that under default `ALL` mode it will still relay other traffic on the same modem settings.

Changing Channels on a Deployed Node

Options for modifying channel config after deployment:

- **Via remote admin** (preferred for remote nodes) - send a channel update from an administrator node whose key is in the remote node's admin-key list (or, on legacy nodes, that shares the admin channel).
- **Via serial/USB** - connect a laptop directly to the node.
- **Via Bluetooth** - only if Bluetooth was left enabled.

Common pitfall: losing the admin keys / admin channel config (or disabling Bluetooth and having no USB access) leaves a remote node inaccessible without a physical site visit. Always back up admin keys and any admin-channel QR codes before deployment.

Advanced Meshtastic Repeater Topics

Store and Forward, position reporting, and telemetry for infrastructure repeater nodes.

Store and Forward

What Is Store and Forward?

Store and Forward (S&F) is a Meshtastic server module that **buffers text messages for nodes that are temporarily offline**. When a client node comes back within range, the S&F server can replay text messages the client missed while it was away.

Important - the buffer is volatile and S&F is not a delivery guarantee. Stored messages live only in the server node's PSRAM. Any reboot, brownout, or power loss on the server node erases ALL stored messages with no recovery. Replay is best-effort, request-driven, and limited by count and time window; it covers text messages only (not telemetry, position, or other packet types). Never treat the buffer as durable storage, and never rely on it for life-safety message assurance - for incident use, assume the buffer can vanish at any moment.

How It Works

1. A node with S&F enabled in *server mode* listens to text messages on its primary channel (channel 0) and stores them in a ring buffer in PSRAM. Note: history retrieval over LoRa is **not available on the default public channel**, so the server must be configured with a non-default primary channel for S&F history to work.
2. When a client node requests history (see Client Configuration below), it sends a replay request.
3. The server transmits buffered messages up to the configured **History Return Max** count and within the **History Return Window** time range.

Hardware Requirements

The S&F server requires an ESP32 device **with PSRAM** - nRF52-based boards cannot run the Store-and-Forward server module at all:

- **Recommended (ESP32-with-PSRAM only):** T-Beam (v1.0 and later), LilyGo T3S3, or other ESP32-S3 devices that include PSRAM.

- **Not supported as an S&F server:** nRF52840-based devices (RAK4631, Nordic development kits) and any board without PSRAM. These cannot act as a Store-and-Forward server.

Server Configuration

1. Navigate to **Config → Module Config → Store and Forward**.
2. Set **Enabled → true**.
3. Set **Is Server → true**.
4. Set **History Return Max** - the maximum number of messages to replay per request (default: 25 messages, which is the value used when this is left at 0).
5. Set **History Return Window** - the time window in minutes from which to replay messages (default 0 = 240 minutes / 4 hours).
6. Save and reboot the node.

Client Configuration

A client node does **not** need the Store and Forward module enabled to request history. To retrieve missed messages:

1. Set role to **CLIENT** (or **CLIENT_MUTE**, labelled "Client Mute" in the app menu, if the device should not rebroadcast).
2. Request history manually: on Android, send a direct message containing **SF** to the server node; on Apple, use the **Client History** option.

History retrieval over LoRa is manually requested as above. Automatic retrieval happens only when an app connects directly to the server node (firmware 2.4+) - it does not happen automatically over the mesh just by coming into range.

Ideal Use Cases

- Hikers or cyclists who move in and out of mesh coverage.
- Vehicles that periodically leave and re-enter a covered area.
- Remote monitoring stations that reconnect on a schedule.
- Community nets that want a convenience replay of recent text traffic - but note S&F is best-effort only and must never be relied on as guaranteed delivery for emergency or life-safety messages.

Limitations

- S&F history retrieval over LoRa is **not available on the default public channel**; you must configure a non-default primary channel to use it. S&F operates only on the primary channel (channel 0) - if your net runs on a private or secondary channel, store-and-forward will not buffer or replay that traffic. Verify replay actually works on YOUR channel before depending on it.
- Text messages only - telemetry, position, and other packet types are not buffered or replayed.
- Large message volumes will cause the ring buffer to overwrite older messages quickly.
- **The buffer lives in PSRAM only.** The server node must be continuously powered and on-mesh; any reboot, brownout, or power loss erases ALL stored messages with no recovery.
- S&F adds overhead to the server node; monitor channel utilisation on high-traffic networks.

Position and Telemetry for Infrastructure Nodes

Why Position Accuracy Matters

An accurate position lets your repeater appear correctly on meshmap.net and in the [Meshtastic app](#)'s node list. (Note that meshmap.net is a third-party map and only shows nodes reporting to the public MQTT server, so it is not an authoritative or complete view of the mesh.) Other operators use your node's reported position to plan coverage, model signal paths, and verify that packets are actually being relayed from the expected location.

Fixed Position vs. GPS

Most unattended infrastructure repeaters do not need GPS hardware. Instead, configure a **fixed position** using the known coordinates of the deployment site:

1. Look up the coordinates of your site with any map application (Google Maps, OSMand, etc.) - right-click the location and copy the lat/lon.
2. In the Meshtastic app: **Config → Position → Fixed Position → Enable**.
3. Enter the **latitude**, **longitude**, and **altitude** (in metres).
4. Save. The node will broadcast this position at the configured interval without needing a GPS fix.

If your device has GPS hardware and the repeater is mobile (e.g. a vehicle relay), leave GPS enabled and set the Smart Position Broadcast threshold appropriate for your speed.

Position Broadcast Interval

This setting controls how often the node announces its location on the mesh.

- **Default:** 15 minutes (with Smart Broadcast enabled). Smart Broadcast increases the frequency when the node is moving.
- **Recommended for static infrastructure:** 1 hour or longer (3600 seconds and up); for a permanently fixed node, many operators use 12 - 24 hours (43200 - 86400 seconds). A repeater that never moves does not need to announce its position frequently.
- Increasing (lengthening) this interval lowers airtime consumption and channel utilisation - important on busy networks.

Smart Position Broadcast

Meshtastic's Smart Position feature broadcasts a new position only when the node has moved beyond a configurable distance threshold. **For a static repeater, disable Smart Position and use a fixed timed interval instead** - Smart Position is designed for moving nodes and may behave unexpectedly on hardware without GPS.

Device Telemetry

Meshtastic can broadcast device health metrics over the mesh. The device metrics are:

- Battery voltage and charge percentage.
- Channel utilisation (ChUtil) - the percentage of time the radio channel is occupied by transmissions (this node's own transmissions plus traffic it senses from others).
- Air utilisation for transmit (AirUtilTX) - the percentage of airtime this node itself spends transmitting.

Enable and configure telemetry under **Module Config → Telemetry**, in the **Device Metrics** section (exact menu labels vary by client and version). Set the broadcast interval - the default of 1800 seconds (30 minutes) is appropriate for most infrastructure nodes; lengthen it to reduce airtime.

Remote Health Monitoring

Once device telemetry is enabled, any operator who can see your node on the mesh can view its reported battery voltage and channel utilisation in the Meshtastic app's node detail view. This provides passive, no-cost status visibility:

- A gradual drop in reported battery voltage warns of a failing solar charge or depleted battery before the node goes offline.

- High channel utilisation (above ~25% ChUtil, or AirUtilTX above ~7 - 8%) indicates congestion - consider lengthening broadcast intervals or relocating the node.
- Uptime resets alert you to unexpected reboots (power glitches, firmware crashes).

Passive telemetry shows health only while the node is alive and in range. It provides **no active alert when the node actually fails** - you simply stop seeing updates, which is easy to miss. For infrastructure you depend on, pair telemetry with an active offline-detection alert (something that notifies you when telemetry stops).

For critical infrastructure nodes, consider setting up a **MQTT bridge** to push telemetry to an external monitoring system - see the Meshtastic MQTT documentation for details. Note that Wi-Fi/MQTT-based monitoring depends on internet connectivity and will fail in exactly the grid-down or internet-out emergencies you may most need it.

Role Configuration and Tuning

Deep-dive configuration of ROUTER, ROUTER_CLIENT, and REPEATER roles including hop limit tuning and fixed-position setup.

ROUTER vs ROUTER_CLIENT vs REPEATER: When to Use Each

Meshtastic's current infrastructure node roles are **ROUTER**, **ROUTER_LATE**, and **REPEATER** (with **CLIENT** covering ordinary nodes, which also relay traffic via managed flooding). Each has distinct behaviors for packet forwarding, position broadcasting, and power management. Choosing the wrong role for your hardware leads to either wasted bandwidth or poor coverage. **Note:** **ROUTER_CLIENT** was *retired in firmware 2.3.15* and is no longer a settable role - it now behaves as **CLIENT**. It is covered below only to explain what to use instead.

Role Behavior Summary

Behavior	ROUTER	ROUTER_LATE	REPEATER
Forwards packets	Yes (high priority)	Yes (rebroadcasts only after others)	Yes (high priority)
Has user-facing interface (BLE/WiFi/Serial)	No (off by default)	No (off by default)	Yes (on by default)
Broadcasts own position	Reduced	Reduced	Never
Appears in node list	Yes	Yes	No (stealth)
Sends telemetry	Reduced	Reduced	None
Preferred for infrastructure	Yes (stationary, well-placed)	Yes (rebroadcast-late nodes)	Yes (privacy-conscious / anonymous relay)

ROUTER

The ROUTER role is designed for dedicated, stationary, well-placed infrastructure nodes. It:

- Rebroadcasts with higher priority ("cuts in line") and always rebroadcasts, even if it hears another node already rebroadcasting - unlike CLIENT, which suppresses its own rebroadcast if a neighbor already did. REPEATER shares this same high-priority always-rebroadcast behavior.
- Automatically enables power-saving sleep (this cannot be turned off) and defaults BLE/WiFi/Serial to OFF, so you don't normally connect an app to it directly. (It can still originate and receive messages at the protocol level - the app connectivity is simply off by default.)
- Reduces position and telemetry broadcasts
- Remains visible in the node list so operators can see it on the map

Use ROUTER when: You have a dedicated always-on, stationary, well-placed infrastructure node (fixed repeater, backbone node) with no human user directly attached. For ordinary nodes, official guidance is to leave them on CLIENT rather than promoting them to ROUTER.

ROUTER_CLIENT (deprecated - use CLIENT or ROUTER instead)

ROUTER_CLIENT was **deprecated and removed in firmware 2.3.15**; it is no longer a settable role and now behaves exactly as CLIENT. It was originally meant for nodes that served double duty:

- A base station that both serves a human operator AND forwards traffic for others
- A vehicle-mounted node that forwards packets while its owner uses it for messaging

Why it was retired: ROUTER_CLIENT nodes created traffic loops and increased channel utilization (consuming hops and causing collisions) when several were deployed in close proximity - the developers have publicly described it as a mistake. Do not treat it as a role you can merely "be cautious with"; it is gone.

What to use instead: For a node that one person uses AND that should also relay for others, leave it on **CLIENT** - CLIENT nodes already rebroadcast via managed flooding while keeping the full client interface. For a dedicated repeater that no one uses interactively, use **ROUTER**.

REPEATER

The REPEATER role is the "silent repeater" option. (Note: REPEATER is a `device.role`; it is **not** "formerly ALL_SKIP_DECODING." ALL_SKIP_DECODING is a value of the separate `device.rebroadcast_mode` setting - the two are independent config axes.) REPEATER:

- Rebroadcasts with high priority but does NOT appear in node lists and does not send NodeInfo - it is anonymous on the mesh. It does not force power-saving sleep (unlike ROUTER).
- Can be combined with rebroadcast mode **ALL_SKIP_DECODING** to forward packets without decrypting them (low CPU overhead). Selecting the REPEATER role alone does **not** enable skip-decoding - you must set `device.rebroadcast_mode ALL_SKIP_DECODING` separately.
- Is not intended for direct messaging (it has no user interface and only responds to other nodes' packets rather than originating messages)
- Does NOT broadcast position or telemetry

Use REPEATER when: You want a completely silent, anonymous infrastructure node - no node-list footprint, no management overhead. Ideal for stealth installations or when you want to avoid cluttering the node list.

Caveat: The REPEATER's invisibility makes it harder to diagnose - you won't see it on the map or in node lists. Use ROUTER instead if you need visibility for network management. (REPEATER is also deprecated as of firmware ~2.7.x; for most new infrastructure, prefer ROUTER or ROUTER_LATE.)

Configuration Commands

```
# Set role to ROUTER (dedicated infrastructure)
meshtastic --set device.role ROUTER

# ROUTER_CLIENT was retired in firmware 2.3.15 - do not use it.
# For a node that is both used and relays, leave it on CLIENT:
meshtastic --set device.role CLIENT

# Set role to REPEATER (anonymous silent relay)
meshtastic --set device.role REPEATER

# (Optional) skip-decoding forwarding is a SEPARATE rebroadcast_mode setting,
# not a role - enable it explicitly if you want it:
meshtastic --set device.rebroadcast_mode ALL_SKIP_DECODING

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

Role Configuration and Tuning

Deep-dive configuration of ROUTER, ROUTER_CLIENT, and REPEATER roles including hop limit tuning and fixed-position setup.

ROUTER vs ROUTER_CLIENT vs REPEATER Role: Deep Dive

Meshtastic provides several device roles for infrastructure nodes that exist to extend network reach rather than serve end users. On current firmware the relevant roles are **ROUTER**, **ROUTER_LATE**, and **REPEATER** (with **CLIENT** being the right choice for the overwhelming majority of ordinary nodes, since CLIENT nodes already rebroadcast via managed flooding). Choosing the wrong role wastes resources, creates unnecessary air-time, or silently breaks the capabilities operators expect. This page dissects each role at the firmware level so you can make an informed decision for every node you deploy.

Important deprecation notice. The **ROUTER_CLIENT** role referenced in this page's title was **retired in firmware 2.3.15** and is no longer a selectable `device.role` value. Do not attempt to set it. If you need a node that both relays and is used interactively by its operator, use **CLIENT** (which already relays) or **ROUTER**. Separately, the **REPEATER** role has itself been **deprecated as of firmware ~2.7.x**; for new infrastructure prefer **ROUTER** or **ROUTER_LATE**. The valid current `device.role` values are CLIENT, CLIENT_MUTE, CLIENT_HIDDEN, ROUTER, ROUTER_LATE, REPEATER, TRACKER, SENSOR, TAK, TAK_TRACKER, and LOST_AND_FOUND. The historical ROUTER_CLIENT material below is retained for context only and clearly marked as legacy.

The Infrastructure Roles at a Glance

Capability	ROUTER	ROUTER_LATE	REPEATER (deprecated)
Rebroadcasts received packets (once, with higher priority than CLIENT roles)	Yes	Yes (rebroadcasts last, after other nodes)	Yes
Power-saving sleep behaviour	Forced on automatically (ESP32); cannot be disabled	Configurable via <code>power.is_power_saving</code>	Does not force sleep by default

Capability	ROUTER	ROUTER_LATE	REPEATER (deprecated)
Visible in the node list / sends NodeInfo	Yes	Yes	No (anonymous relay, sends no NodeInfo)
App connectivity (BLE / WiFi / Serial) by default	Off by default	On (client-style)	Off
Relative overhead	Standard	Standard	Minimal overhead (per docs)

All of ROUTER, ROUTER_LATE, and REPEATER rebroadcast each packet *once* with higher priority than ordinary CLIENT nodes. The retired ROUTER_CLIENT role has been omitted from this table; it is not settable on current firmware.

ROUTER Role

A **ROUTER** node is the canonical fixed-infrastructure role, designed for stationary, well-placed nodes. When you set a node to ROUTER it adjusts its behavior in the following ways:

- ROUTER automatically enables power-saving sleep (ESP32) via `power.is_power_saving`, and this **cannot be turned off**. The LoRa radio stays in standby to wake on incoming packets — it is not a node that ignores power management.
- It defaults BLE / WiFi / Serial connectivity **off**, so you don't normally connect to it to text directly.
- Received packets are rebroadcast once, with higher priority than CLIENT roles, subject to duplicate-detection and the hop counter.
- The device **is visible in the node list** and sends its own `NodeInfo` (and position, if configured) so other nodes know it exists and can display it on the map.
- A ROUTER relays standard packets, including traceroute responses, like any non-mute node. (NeighborInfo is a separate, optional module any role can enable — it is not a defining trait of the ROUTER role.)
- A ROUTER *can* still originate and receive messages at the protocol level. What's true is that its app connectivity (BLE/WiFi/Serial) is off by default, so in current firmware it does not present an interactive messaging UI and you typically won't have an app connected to send from it. This is a connectivity default, not a removed "Send" feature — the firmware does not omit the send code path. If interactive messaging at the site is needed, use CLIENT or ROUTER_LATE rather than ROUTER.

Setting the ROUTER role via CLI

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

Verifying the role was applied

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

The `--get` verb is the standard counterpart to `--set` for reading any config field in the Python CLI.

ROUTER_LATE Role

ROUTER_LATE is the modern infrastructure role for a node that should rebroadcast *only after* other nodes have had a chance to, rather than with top priority. It is useful within a local cluster where you want a relay that fills gaps without dominating airtime. Unlike ROUTER, it does not force power-saving sleep on, and it keeps client-style app connectivity available. For most stationary backbone infrastructure prefer ROUTER; choose ROUTER_LATE for nodes that should defer to the rest of the mesh first.

ROUTER_CLIENT Role (retired — legacy reference only)

ROUTER_CLIENT was retired in firmware 2.3.15 and is not a settable role on current firmware. Historically it was described as a "superset of ROUTER" that retained ROUTER's relay behaviour while also letting the node originate messages as a client. This is preserved here only for operators encountering older documentation or very old firmware:

- It was described as allowing text messages to be composed and sent from the device. On current firmware, if you need a relay that an operator can also message from, use **CLIENT** (which relays via managed flooding).
- It was described as broadcasting position from the device's own GPS or configured coordinates. That position-broadcast behaviour now applies to CLIENT-family roles.
- It was promoted for an operator who wanted to use the node interactively during events or emergency activations without compromising the relay function. Today, use a **CLIENT** node for that. (Note: Meshtastic carries text, position, and telemetry — it does not carry voice — so "voice-channel communication" was never accurate terminology.)

Do not use the legacy command below on current firmware — it will fail because ROUTER_CLIENT is no longer an accepted `device.role` value. For a relay that an operator also uses interactively, set `device.role CLIENT` instead.

Legacy ROUTER_CLIENT CLI (no longer valid)

```
# Retired in firmware 2.3.15 – this command fails on current firmware.  
# Use:  meshtastic --set device.role CLIENT  
# (legacy:  meshtastic --set device.role ROUTER_CLIENT)
```

REPEATER Role (deprecated)

The **REPEATER** role is the most minimal relay option, designed for nodes that should retransmit packets with minimal overhead and produce no broadcasts of their own. **Note: REPEATER was deprecated as of firmware ~2.7.x; for new infrastructure consider ROUTER or ROUTER_LATE.** Its documented behaviour:

- Received packets are rebroadcast once, with higher priority than CLIENT roles — similar to ROUTER in the act of relaying.
- It is **not visible in the node list** and does *not* send `NodeInfo` — it is effectively anonymous on the mesh. It will not appear in the node list of clients that have not heard it directly.
- It does not originate its own broadcasts (telemetry, position, `NodeInfo`); per the docs it "only responds to other nodes' packets instead of originating messages."
- Per the docs it rebroadcasts with minimal overhead. (It does *not* force power-saving sleep, unlike ROUTER.)

The anonymity of the REPEATER role is a deliberate design choice: relay nodes that aren't individually addressed produce less management overhead on the mesh. Because a REPEATER is not in the node list, you generally cannot select or ping it by node ID in the app the way you can a ROUTER.

Setting the REPEATER role via CLI

```
meshtastic --set device.role REPEATER
```

Hop Count Handling Across Roles

All relaying roles participate in hop-count decrement identically. When a packet arrives with a remaining hop count of N, the forwarding node decrements it to N-1 before rebroadcasting. If N is already 0, the packet is consumed locally but not forwarded. ROUTER and REPEATER have rebroadcast *priority* (they may rebroadcast sooner, and defer less to other rebroadcasters) but they still decrement and honour `hop_limit` — there is no hop-counter bypass or "always forward regardless" mode.

The practical consequence is that placing a relay mid-path does use a hop. Default `hop_limit` is **3** and the maximum is **7**; "really, 3 is fine" per the docs. Ensure the number of relay hops between any two endpoints does not exceed your configured `hop_limit`. Raising `hop_limit` to "fix" dropped messages can worsen congestion (more airtime and collisions), so pair any increase with a check on channel utilisation.

Choosing the Right Role

Community fixed repeater on a hilltop or tower

Use **ROUTER**. The node should be visible to the community (appears in node list, sends NodeInfo, relays traceroutes) but should not generate user traffic of its own. Operators can still access the serial console locally for configuration. For a node that should defer to the rest of the mesh before rebroadcasting, consider **ROUTER_LATE**.

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

Ham operator's home station that also relays

Use **CLIENT**. CLIENT nodes already rebroadcast via managed flooding, so a home station set to CLIENT both relays and lets you participate in mesh conversations — sending alerts, coordinating with your community, or running net check-ins from the same hardware. (The old advice to use ROUTER_CLIENT here is obsolete; ROUTER_CLIENT was retired in 2.3.15.)

```
meshtastic --set device.role CLIENT
```

Minimal-overhead anonymous relay

Historically **REPEATER** was the choice for a small node tucked into a building to bridge two otherwise disconnected areas with no topology visibility. Because REPEATER is deprecated as of firmware ~2.7.x, prefer **ROUTER** (or **ROUTER_LATE**) for new deployments; only use REPEATER if you specifically need an anonymous relay and understand it is being phased out.

```
meshtastic --set device.role ROUTER    # preferred; REPEATER is deprecated
```

Power Consumption Comparison

Idle power profiles differ by role. A **REPEATER** keeps the radio active and does not force sleep, whereas a **ROUTER** uses forced power-saving sleep (ESP32) that cannot be disabled — so their idle draw is not the same. The differences otherwise arise from background processing:

- **REPEATER** has minimal overhead per the official docs, because it skips originating broadcasts and node-presence maintenance. (We do not cite a specific kilobyte heap figure, as no firmware source quantifies it.)
- **ROUTER** forces power-saving sleep on (ESP32); **ROUTER_LATE** leaves sleep configurable via `power.is_power_saving`. If a GPS is active, GPS module current (which varies widely by module — consult the module's datasheet) can dominate other draw.
- LoRa TX current is board- and TX-power-dependent. For an SX1262-class radio alone, TX is roughly ~118 mA at +22 dBm and RX is roughly ~5 mA (Semtech SX1262 datasheet); boards with an external PA draw more, and total board draw (including the MCU) is higher than the radio alone. Treat any single current figure as approximate and board-dependent.
- Power-saving sleep is broadly available via `power.is_power_saving` to most roles (all except TRACKER and SENSOR). It is **not** exclusive to CLIENT: ROUTER forces it on automatically, while REPEATER does not sleep by default.

Quick CLI power-related settings for infrastructure nodes

```
# Disable Bluetooth to save power (figure is approximate; verify against the Bluetooth config docs)
```

```
meshtastic --set bluetooth.enabled false
```

```
# Reduce the screen on-time to save power. Note: the documented field is
```

```
display.screen_on_secs.
```

```
# A value of 0 is the 10-minute DEFAULT, not "off"; set a small nonzero value to dim sooner.
```

```
meshtastic --set display.screen_on_secs 10
```

```
# Set a lower TX power only if nodes are nearby (reduces TX draw and channel utilisation).
```

```
# tx_power 0 is the default and means "use the region-legal maximum." Never set a higher fixed
```

```
# value than your region/antenna combination legally allows (note the >6 dBi gain-reduction  
rule).
```

```
meshtastic --set lora.tx_power 17
```

Hop Limit Configuration for Repeaters

The hop limit is one of the most important and most misunderstood parameters in a Meshtastic mesh. Setting it correctly reduces unnecessary rebroadcasts, controls how far a message propagates, and prevents broadcast storms that saturate the channel. This page explains exactly what `hop_limit` does, when to change it, and the CLI commands to apply it.

What hop_limit Controls

Every Meshtastic packet carries a **hop count field** in its header. This field is initialised to the `hop_limit` value configured on the *originating* node at the time the packet is sent. Each relay node (ROUTER, ROUTER_LATE, or REPEATER) decrements this field by 1 before rebroadcasting. (The older ROUTER_CLIENT role was deprecated and removed in firmware 2.3.15 — use ROUTER or ROUTER_LATE instead; REPEATER is also deprecated as of firmware ~2.7.x.) When a node receives a packet with a hop count of 0 it delivers the packet locally but does *not* forward it further. (See the Meshtastic mesh algorithm docs: "If any mesh node sees a packet with a HopLimit other than zero, it will decrement that HopLimit and attempt to rebroadcast on behalf of the original sending node.")

In a simple linear chain, the maximum relay chain length is:

```
Maximum relaying nodes = hop_limit
```

```
Nodes reached along an idealised single chain = hop_limit + 1 (originator counts as hop 0)
```

Note: real meshes are not linear chains. Many nodes can hear each hop, so the total number of nodes that actually hear a packet is topology-dependent — the formula above describes an idealised single chain, not a fixed total node count.

With the default `hop_limit = 3`, a packet originating at node A can reach:

- All nodes within direct radio range of A (hop 0 -> 1)
- All nodes reachable via one relay (hop 1 -> 2)
- All nodes reachable via two relays (hop 2 -> 3)

- Once a packet's remaining HopLimit reaches 0 it is delivered locally but not rebroadcast further (hop 3 -> consumed, not forwarded)

In practice the default of 3 covers most community-sized networks with reasonable relay density.

The Default: hop_limit = 3

As of current firmware (2026), the firmware default is 3 — a careful balance. It is high enough to traverse a typical multi-node mesh with a few infrastructure repeaters, and low enough that a single rogue or misconfigured node cannot cause runaway rebroadcasting. (This is a firmware-version-dependent default; verify against your installed firmware.) Leave it at 3 unless you have a specific and measured reason to change it.

When to Increase the Hop Limit

Large geographically spread networks

If your community network spans a large geographic area - for example a county-wide emergency communications mesh with repeaters spaced 20 - 30 km apart - a hop count of 3 may not be sufficient to bridge the entire path. In this case, increasing to 4 or 5 gives messages the relay budget to traverse more infrastructure nodes. Bear in mind that raising hop_limit increases airtime and channel utilisation; prefer adding infrastructure to shorten paths where you can.

Before increasing, first verify the actual hop count needed by running a traceroute between the two furthest nodes:

```
meshtastic --traceroute '!abcd1234'
```

Count the intermediate hops in the output. Set `hop_limit` to at least that number plus one for margin.

```
# Increase hop limit on the originating/infrastructure node
meshtastic --set lora.hop_limit 4
```

When to Decrease the Hop Limit

Small, dense urban networks

In a dense neighbourhood deployment where every node can hear at least 2 - 3 others directly, a hop limit of 3 generates significant redundant retransmissions. Consider reducing to 2 if traceroutes show no path requires more than 2 relays.

```
meshtastic --set lora.hop_limit 2
```

Local-only repeater that should not propagate far

Important: setting `hop_limit` low on a repeater does **NOT** limit the packets it relays for others — it only limits how far the repeater's *own* messages (its NodeInfo, position, telemetry) travel. To limit how far transit traffic propagates through your infrastructure you must coordinate settings network-wide (see below) or use rebroadcast-mode / role settings.

If you deploy a repeater specifically to bridge two buildings on the same campus - not to reach the wider regional mesh - you can reduce the reach of the traffic *it itself originates* by setting `hop_limit = 2` on its own packets (its own NodeInfo, position, telemetry). This does **not** prevent it from forwarding transit packets that arrive with a remaining count of 3, so it does not constrain its repeater function — it only limits the blast radius of traffic the repeater itself generates.

Note: the `lora.ignore_incoming` array is a **blocklist** — node IDs listed there have their packets dropped on receive — not a whitelist of nodes to serve. It is also deprecated in the client apps. To ignore a specific node, long-press it in the app and mark it ignored rather than relying on this setting.

The Broadcast Storm Risk From High Hop Counts

The maximum hop_limit is 7 (it cannot be set higher). Values of 6 and 7 are valid settings, not a forbidden range — but the Meshtastic project strongly recommends leaving hop_limit at 3 unless you have a specific, verified reason to raise it, because unnecessarily high hop counts cause

network problems. Here is why:

- High hop limits in dense networks multiply rebroadcasts and waste airtime. Meshtastic's managed flooding suppresses duplicate (already-heard) rebroadcasts, which bounds the effect well below a naive geometric explosion — but it mitigates rather than eliminates the congestion. Keep `hop_limit` as low as your path requirements allow.
- LoRa is a half-duplex medium. Each retransmission blocks all other transmissions in range for the duration of the air-time. Airtime per packet is payload-dependent: a long packet on the Long Slow preset (SF12, 125 kHz, ~180 bps data rate) can occupy the channel for roughly 2 - 3 seconds. Combined with high hop counts in a dense mesh this can push channel utilisation very high and severely degrade the network.
- Meshtastic **does** have a back-off mechanism: it uses CSMA/CA (similar to WiFi), not Ethernet's CSMA/CD. All transmitters perform Channel Activity Detection (CAD) before transmitting, and if the channel is busy they wait. The amount of wait is randomly picked from a contention window whose size grows with channel utilisation, specifically to limit collisions during congestion.

If you feel you need a high hop limit, the better solution is usually to add more infrastructure repeaters to shorten the required path, rather than raising the hop counter.

Configuring `hop_limit` via CLI

```
# Set hop limit to 3 (default - recommended for most networks)
meshtastic --set lora.hop_limit 3

# Set hop limit to 4 (for large spread-out networks after traceroute verification)
meshtastic --set lora.hop_limit 4

# Set hop limit to 2 (for small dense networks or local-only repeaters)
meshtastic --set lora.hop_limit 2

# Verify the applied value
meshtastic --get lora.hop_limit
```

`hop_limit` Is a Per-Node Origination Setting

An important subtlety: `hop_limit` controls the initial value placed in packets that *this node originates*. It has no effect on packets that arrive from another node already carrying a hop count - those are forwarded as received (after decrement). Therefore:

- Setting `hop_limit = 2` on a relay node limits only that node's own originations.
 - A packet originating elsewhere with `hop_limit = 5` will still traverse your relay with a remaining count of 4 after decrement - your local setting does not cap it.
 - To limit how far foreign traffic propagates through your infrastructure, you must coordinate `hop_limit` settings across all nodes in your deployment, or use channel-level policies.
-

Monitoring Channel Utilisation After Changes

After adjusting `hop_limit`, monitor channel utilisation via the [Meshtastic app](#) or web UI, where it is clearly labelled in the node detail view. Channel utilisation (ChUtil) is measured over a 1-minute window (six 10-second periods). Use these health bands consistently: green below 25%, orange 25 - 50%, red above 50% ChUtil; AirUtilTX should be much lower (a few percent). Note that ChUtil is a lagging rolling average, so it can read healthy while the channel is momentarily saturated. If you see utilisation spike after a `hop_limit` increase, your network density may not support the change and you should revert.

Fixed Position for Repeater Nodes

A repeater node that knows its own location serves the community in two ways: it appears accurately on coverage maps, and it lets neighbouring nodes calibrate their own position estimates. Without a fixed position, a GPS-less repeater either appears at coordinate (0, 0) in the ocean or does not appear on the map at all. This page explains how to configure a precise static position, how to reduce position precision for privacy, and how to minimise the air-time cost of periodic position broadcasts.

Why a Fixed Position Matters for Infrastructure Nodes

- **Coverage visibility** - Community members and emergency coordinators use the network map to understand what areas are served. A repeater with an accurate position lets them trace coverage boundaries and identify gaps.
 - **Traceroute path correlation** - When operators run traceroutes, hop positions appear on the map. A correctly placed repeater gives a geographically meaningful path diagram.
 - **SNR context** - Signal-to-noise ratio reports are much more useful when the receiving node's coordinates are known, because operators can correlate signal strength with distance and terrain.
 - **No GPS module required** - Many dedicated repeater platforms (e.g. RAK WisBlock with no GPS module, T-Beam with GPS disabled) can broadcast a fixed configured position without incurring GPS power draw.
-

Configuring a Static GPS Position Without a GPS Module

Method 1: Meshtastic CLI (recommended)

The `--setlat`, `--setlon`, and `--setalt` flags write a fixed position directly to the device's configuration storage. Once set, this position is broadcast as the node's location even with no GPS hardware present. This is the documented, recommended way to set a fixed position.

```
# Set position for a repeater at the top of Mount Davidson, San Francisco
# Latitude: 37.7406, Longitude: -122.4538, Altitude: 282 metres
meshtastic --setlat 37.7406 --setlon -122.4538 --setalt 282
```

Determine the coordinates of your site using Google Maps, CalTopo, or any mapping tool that provides WGS84 decimal degrees (the standard Meshtastic expects). Right-click the exact antenna location on Google Maps to copy coordinates.

Method 2: Meshtastic Python API

If you prefer scripting, the Python API exposes `setFixedPosition()` on the local node, which takes decimal-degree latitude and longitude plus an integer altitude in metres:

```
from meshtastic.serial_interface import SerialInterface

iface = SerialInterface()

# Set a fixed position: latitude, longitude (decimal degrees), altitude (metres)
iface.localNode.setFixedPosition(37.7406, -122.4538, 282)

iface.close()
```

Method 3: Meshtastic Web UI

Connect to the node's web interface (available on ESP32-based devices at `http://meshtastic.local` or the device's IP when connected to Wi-Fi). Navigate to **Config → Position**, enable *Fixed Position*, and enter latitude, longitude, and altitude. Save and reboot.

Position Precision: Reducing Exact Location for Privacy

If the repeater is on private property and the owner does not want the exact address broadcast on the mesh, reduce position precision. Position precision is a **per-channel** setting (`module_settings.position_precision`), so it is configured against a specific channel index rather than globally. Meshtastic supports precision levels from full resolution (sub-metre) down to a heavily-rounded approximation. The exact bits-to-radius mapping is defined by the in-app precision slider; the values below are approximate and may shift between firmware versions - confirm against the current Meshtastic position-config docs.

Common choices for community repeaters:

- **Full precision** (default) - broadcast exact coordinates. Acceptable for tower sites, mountain tops, and public-land installations.
- **~1.5 km precision** - rounds to roughly a 1.5 km radius. Identifies the general neighbourhood without revealing the specific building.
- **~12 km precision** - rounds to roughly a 11.7 km radius. Useful for regional-scale maps where sub-kilometre accuracy is unnecessary and the operator prioritises privacy.

```
# Position precision is per-channel; --ch-index selects the channel (0 = primary)
# Set position precision bits to 14 (approximately a 1.5 km radius)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 14

# Set position precision bits to 11 (approximately a 11.7 km radius)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 11

# Restore full precision (precision bits 32)
meshtastic --ch-index 0 --ch-set module_settings.position_precision 32
```

The `position_precision` value is a bit-field width - higher values mean more significant bits retained from the raw coordinate, i.e. higher accuracy. Values below about 10 round so aggressively that the position becomes almost meaningless for map purposes. Because this is a per-channel setting, you can carry different precision on different channels.

Position Broadcast Interval for Fixed Nodes

A fixed repeater does not move. By default the node broadcasts its position on a periodic interval (the firmware default is 15 minutes), and smart position broadcast - controlled by `position.position_broadcast_smart_enabled` (default `true`) - further reduces transmissions when the node is stationary. For a static infrastructure node you can simply lengthen the interval to at least 12 hours (43200 seconds), or even 24 hours (86400 seconds), to avoid wasting air-time that could be used by mobile nodes.

```
# Set position broadcast interval to 12 hours (43200 seconds)
meshtastic --set position.position_broadcast_secs 43200

# Set position broadcast interval to 24 hours (86400 seconds) - recommended for fixed sites
meshtastic --set position.position_broadcast_secs 86400

# Smart position broadcast is on by default; set false only if you want to disable it
meshtastic --set position.position_broadcast_smart_enabled false
```

With an 86400-second interval, a fixed node broadcasts its position approximately once per day (plus once at boot). The firmware default interval is 15 minutes; increasing it to the 12-24 h range for a static node saves many transmissions per day across a network with multiple infrastructure nodes.

Verifying the Fixed Position Was Applied

```
# Check position configuration on the device
meshtastic --get position

# Pull the device info including last known position
meshtastic --info | grep -A 5 position
```

After the node reboots following a position update, connect to the [Meshtastic app](#) and look at the node list. The repeater should appear at the correct location on the map within one position

broadcast interval.

Altitude Accuracy

Altitude in Meshtastic is stored in whole metres. It is commonly referenced to the WGS84 ellipsoid rather than mean sea level, but the `Position` message also carries an `ALTITUDE_MSL` flag that indicates when a value is given relative to mean sea level - so do not assume the value is always ellipsoidal. For antenna-height accuracy, use the elevation of the antenna itself, not ground level at the base of the tower. For most community mapping purposes, ground-level elevation from a topo map is sufficient - the difference rarely affects coverage visualisations.

Find accurate elevation using the USGS National Map (`apps.nationalmap.gov`) or `open-elevation.com/api/v1/lookup?locations=LAT,LON` for non-US sites.

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.

Meshtastic Repeater Network Patterns

Designing for Reliability: N+1 Redundancy

Designing for Reliability: N+1 Redundancy

In plain terms: make sure that no single node failure can split your network into two halves that can no longer reach each other. A mesh network is only as reliable as its weakest single point of failure. In graph theory terms, a node whose removal splits a connected graph into two or more disconnected components is called a cut vertex. Real-world Meshtastic deployments often develop cut vertices without operators realizing it - especially as networks grow organically. N+1 redundancy means ensuring that for every critical backbone node, at least one backup path exists so that the loss of any single node does not partition the network.

Important caveat: a second node at the SAME site protects only against single-device failure (firmware crash, board death). It does NOT protect against the failures most likely in an emergency - site power loss, lightning, tower collapse, fire, flooding - because both nodes share that fate. For incident-grade redundancy, the backup MUST be at a separate site on independent power.

Identifying Critical Backbone Nodes

Start by drawing your network on paper or a whiteboard. Place each node as a dot and draw lines between nodes that can hear each other directly. Now ask: if I erase this dot, does the network split into two disconnected groups? Any node where the answer is yes is a cut vertex and a single point of failure.

For larger networks, the most reliable analysis is manual: run `meshtastic --tracroute` between nodes on opposite sides of a suspect backbone node - if the only path routes through that one node, it is critical. Third-party visualizers such as meshmap.net can help you eyeball topology, but they only show nodes that report to the public MQTT server (a gateway is required) and are not authoritative; do not rely on them as a complete picture of your mesh.

Providing Backup Paths

Once critical nodes are identified, the fix is straightforward in concept: ensure each one has a backup. Two approaches work well in practice:

- **Second node at the same site:** Place a second ROUTER node at the same location as the critical node. Both nodes cover the same area, so if one fails at the device level, the other continues forwarding. This costs hardware but is simple to maintain. Be aware of two limits: (1) two co-located relays both rebroadcasting add airtime and can raise channel utilization in that area - prefer one active ROUTER plus a standby, or stagger roles, and reconcile this against the channel-utilization guidance; and (2) as noted above, a same-site pair does NOT protect against site-level failures.
- **Alternate mesh path:** Add a node at a different location that bridges the same two network segments. This is more work to plan but is far more robust - it protects against site-level failures (power outage, lightning, physical damage) rather than just single-node failures. This is the approach required for incident-grade redundancy.

Testing Redundancy

Testing is non-negotiable. Make sure no single node failure splits your network: a backup path that exists on paper but has never been verified may fail in practice due to marginal signal, wrong node roles, or misconfigured hop limits. The test procedure is simple:

1. Identify two nodes on opposite sides of the critical backbone node you want to test.
2. Run a traceroute between them and record the path.
3. Briefly power the backbone node down to simulate failure. (Power it off - do NOT simply unplug its antenna while it is transmitting; transmitting into a disconnected antenna can damage the radio.)
4. Run the traceroute again. The path should route around the powered-down node.
5. If routing fails, the backup path is insufficient and must be improved before the critical node is trusted.

A passing reroute test proves the backup path *exists*, not that it has *capacity*. Under incident load the backup path carries the failed node's traffic on top of its own and may saturate. Verify the backup path also has channel-utilization headroom for surge traffic, not just connectivity.

Test redundancy at least quarterly, and ALWAYS as part of pre-incident readiness checks or before any planned event you expect to rely on the network. Annual testing is insufficient for networks used in emergencies - paths degrade silently between tests, and any time the physical environment changes significantly you should retest.

Network Mapping Tools

Third-party visualizers such as meshmap.net provide a visual overlay of node positions for nodes that report to the public MQTT server, which can help you spot obvious topological bottlenecks - but they are not authoritative and will not show nodes that lack an internet gateway. The ground truth for redundancy verification is the `meshtastic --traceroute <destination_id>` CLI command, which reveals the actual hop-by-hop path packets take in real time.

Planned Maintenance Procedures for Live Networks

Planned Maintenance Procedures for Live Networks

Taking a backbone node offline for maintenance - whether for firmware updates, hardware replacement, or antenna adjustments - affects the users routing through it. With proper planning, that impact can be reduced to a brief interruption rather than a prolonged outage. This page describes a repeatable maintenance procedure for Meshtastic backbone nodes in active community networks.

Pre-Maintenance Checklist

Complete these steps before any planned maintenance window:

- **Notify the community:** Post advance notice (24-48 hours) in your community communication channels - Discord, Signal group, or whatever your community uses. Include the node name, scheduled time, and estimated duration. Users who depend on that backbone link can plan accordingly.
- **Verify backup paths:** Run traceroutes from nodes on either side of the target to confirm alternate routing exists. **If no backup path exists, do NOT take the node offline for non-emergency maintenance until a temporary relay is deployed and verified, or defer the maintenance.** For a sole-path backbone node, planned maintenance without a backup path is itself a planned outage - schedule it only when the network is not needed and notify all users that the network will be DOWN, not merely degraded. Always have a rollback plan: a failed firmware flash can leave a node crash-looping, so never update or remove the only repeater on a path without a verified way to restore it.
- **Schedule during low-traffic hours:** 2-5 AM local time is typically the quietest window for community mesh networks. Emergency networks may have different quiet windows -

check your message logs to identify the lowest-traffic period.

- **Document current configuration:** If replacing hardware, record all node settings (node name, channel configuration, role, hop limit) so the replacement can be configured identically before going live.

During Maintenance

Where possible, power down gracefully rather than performing a hard power-off. A graceful shutdown lets the node stop transmitting cleanly and avoids cutting off a packet mid-transmission or corrupting the node's stored configuration during a flash or write. For solar-powered nodes, the practical approach is to disconnect the load output of the charge controller rather than physically unplugging the node in the dark.

Perform the maintenance task - firmware flash, hardware swap, antenna replacement - as quickly as practical. Every additional minute of downtime increases the chance of a user encountering a failed message delivery.

Post-Maintenance Verification

Before declaring the node returned to service, verify it from multiple directions:

1. Send a test message from a node that previously routed through the maintained node and confirm delivery.
2. Run traceroutes from multiple directions to confirm the node is routing normally.
3. If you run a monitoring system (for example MQTT/Grafana or a third-party map such as meshmap.net), confirm it shows the node online AND recently active. Note that seeing a node online is necessary but NOT sufficient - a node can appear up while failing to relay (for example, wrong role or rebroadcast mode). Always also complete step 1 (a real test message routed through the node) before declaring it back in service.
4. Update your network maintenance log with the date, work performed, and any configuration changes.

Emergency Rollback Procedure

If a replacement node does not work as expected - wrong firmware, hardware fault, or configuration error - act quickly. Restore the original node if it is still functional, or connect a known-good spare configured to match the original settings. If neither is possible, notify the community immediately that the outage is extended and provide an estimated restoration time. For any network used in emergencies, keep at least one spare node per critical site, PRE-CONFIGURED to match that site (role, channel/PSK, hop limit, fixed position). An unconfigured spare is not a substitute - configuring on-site under incident pressure wastes the very time the spare was meant to save.

After any unplanned extended outage, conduct a brief post-incident review: what failed, why, and what process change would prevent recurrence. Even a one-paragraph note in the maintenance log is valuable for future operators.

Channel Utilization Management

Channel Utilization Management

Channel utilization (CU) is one of the most important health metrics for a Meshtastic network, yet it is frequently misunderstood or ignored until problems become severe. Understanding what CU measures, what causes it to rise, and how to bring it back down is essential knowledge for any network operator running more than a handful of nodes.

What Is Channel Utilization?

Channel utilization is the percentage of time the radio channel is occupied by transmissions, measured over a rolling **1-minute window**. Meshtastic breaks the minute into six 10-second sub-windows and sums the milliseconds of airtime used in each. A CU of 10% therefore means the channel was busy for roughly 6 seconds out of the last minute (not 90 seconds out of 15 minutes). Meshtastic calculates CU locally on each node by monitoring the time its own radio is keyed up, plus time spent sensing channel activity from other nodes (via Channel Activity Detection). Note that CU counts the airtime of *all* LoRa traffic heard on the same modem settings, including non-Meshtastic LoRa. The reported CU figure is visible in the [Meshtastic app](#) under the node detail view and in the device telemetry metrics channel. (Do not confuse this 1-minute window with the EU 10% duty-cycle limit, which is measured over a 1-hour window.)

Healthy, Warning, and Critical Thresholds

These bands match the color coding used by the Meshtastic apps:

- **Under 25% (green):** Healthy. The channel has ample headroom for message traffic, routing overhead, and telemetry. Packet loss due to collisions is rare.
- **25-50% (orange):** Warning zone. You may begin seeing occasional packet loss, especially during bursts of activity. Investigate the causes and plan remediation before the situation worsens.
- **Over 50% (red):** Critical. At this level, the probability of two nodes transmitting simultaneously and causing a collision is high enough to cause significant packet loss on a

sustained basis. User-visible symptoms include messages that appear to send but are not received, slow acknowledgements, and missed position updates.

If you prefer a stricter operational target for infrastructure (for example, keeping a busy backbone channel below ~20-25% to preserve headroom for incident traffic), treat that as a conservative operator guideline, not the system threshold. CU is also a lagging rolling average: a channel can briefly saturate while the reported figure still looks healthy.

Common Causes of High Channel Utilization

Several factors drive CU up. Understanding which ones apply to your network guides the correct fix:

- **Too many ROUTER nodes in close proximity:** ROUTER (and REPEATER) nodes always rebroadcast packets they hear, and in a dense cluster those retransmissions pile on top of each other. As an illustration, several co-located ROUTERS can multiply retransmission traffic without providing a proportional coverage benefit. Managed flooding (listen-before-rebroadcast plus duplicate detection) partially mitigates this, so the multiplier is approximate rather than an exact law, but co-located always-rebroadcasting ROUTERS still inflate traffic and collision rates.
- **Hop limit too high for network geography:** A hop_limit of 5 in a network where the farthest node is only 2 hops from the origin means packets are retransmitted up to 5 times unnecessarily. Every extra retransmission is wasted airtime.
- **High message traffic:** Active communities that send many messages, combined with frequent position and telemetry updates from many nodes, can saturate even a well-configured channel.
- **Routing loops:** Meshtastic actively suppresses persistent routing loops via hop-limit decrement and duplicate detection, so true loops are uncommon. If a node appears twice in a traceroute path, treat it as an unusual symptom worth investigating (asymmetric links or misconfiguration) rather than a routine loop indicator.

Remediation Strategies

These remediations are proactive. They must be applied *before* an incident. Once a channel saturates during an event, Meshtastic has no automatic congestion recovery; the only mid-incident fix is to reduce traffic (fewer senders, shorter messages, suppress telemetry), which requires discipline from every operator rather than a single config change - and reconfiguring deployed nodes mid-incident is often impossible. Provision capacity headroom in advance.

- **Reduce hop_limit:** Set hop_limit to the minimum value that still delivers packets to all nodes in your network. For most community deployments, three to four hops is sufficient.

- **Convert excess ROUTER nodes to CLIENT_MUTE or CLIENT:** Audit your node roles. Nodes in the same area do not all need to be ROUTERs. Designate one or two strategically placed nodes as ROUTER and set the rest to CLIENT or CLIENT_MUTE to suppress retransmissions from densely packed areas. (CLIENT still rebroadcasts under managed flooding; CLIENT_MUTE does not rebroadcast at all.)
- **Separate co-located networks by frequency slot:** Two networks on different logical channel numbers but the same modem preset still share the same RF frequency and will still collide. To actually separate them in RF you must change the LoRa frequency slot (or the modem preset / region settings), not just the logical channel index.
- **Consider MeshCore for large networks:** Meshtastic uses managed flooding for broadcasts - nodes listen first and suppress their rebroadcast if a neighbor already relayed the packet, while ROUTER/REPEATER nodes always rebroadcast (since v2.6, direct messages use next-hop routing rather than flooding). In networks with dozens of active nodes and high message traffic, broadcast flooding is a major driver of high CU. MeshCore uses path discovery with source/next-hop routing - it floods once to discover or recover a route, then sends future messages along the learned path - which can scale better in large, dense, high-traffic community networks. Consult the official MeshCore documentation for its exact routing model.