Meshtastic Configuration Reference

Comprehensive reference for all settings under Config > Device, Config > Position, and Config > Network in the Meshtastic firmware.

- Device Configuration Settings
- Position Configuration Settings
- Network Configuration Settings

Device Configuration Settings

The Device configuration section (**Config > Device**) controls fundamental node behavior: its role in the network, how it handles rebroadcasting, node info broadcasts, and administrative access. These are the most impactful settings for network performance and should be understood before deploying any node.

Access these settings in the Meshtastic app under **Settings > Radio Configuration > Device**, via the web interface at `http://<node-ip>/config`, or through the Python CLI with `meshtastic --set device.*`.

Role

Config key: `device.role`
Default: `CLIENT`

The Role setting is the single most important Device configuration choice. It tells the firmware how this node should behave in the mesh - specifically, how aggressively it should rebroadcast packets, whether it should send periodic node info, and how it prioritizes battery life versus network contribution.

Available Roles

CLIENT

The standard role for personal devices carried by users. CLIENT nodes:

- Participate normally in mesh routing (rebroadcast packets they receive)
- Send NodeInfo broadcasts on the configured interval
- Receive and send messages on all subscribed channels
- Display in other nodes' neighbor lists

Use when: You are deploying a personal handheld node, a node you carry daily, or any general-purpose device.

CLIENT_MUTE

Identical to CLIENT but with packet rebroadcasting disabled. A CLIENT_MUTE node receives packets but does not relay them to other nodes. It effectively becomes a receive-only listener on the mesh routing layer.

Use when: You have many devices in close proximity (e.g., all members of a team within radio range of each other). Adding non-muted CLIENT nodes in a dense cluster creates redundant rebroadcasts that waste airtime. Making most of them CLIENT_MUTE reduces channel utilization while maintaining full message delivery to those devices.

Also useful for: Devices behind NAT or VPN tunnels acting as monitoring receivers; test devices you don't want to affect mesh traffic.

ROUTER

Designed for fixed infrastructure nodes at high elevation or central locations. ROUTER nodes:

- Rebroadcast packets with the highest priority
- Use a longer NodeInfo broadcast interval to reduce overhead traffic
- Are favored in routing decisions - the mesh prefers to route through ROUTER nodes
- Disable the LED activity indicator and screen (if present) by default to save power

Use when: Deploying a node on a rooftop, tower, hilltop, or any other fixed elevated location whose primary job is to relay traffic for other nodes. ROUTERs form the backbone of a community mesh.

Important: Only set ROUTER on nodes that will genuinely be at good RF locations. A ROUTER with poor antenna placement will be preferred for routing but perform poorly, degrading the network. It's better to use CLIENT on a node that isn't actually a good relay point.

ROUTER_CLIENT (deprecated; use ROUTER or REPEATER instead)

A hybrid role combining ROUTER routing behavior with CLIENT-level NodeInfo and message participation. ROUTER_CLIENT (deprecated; use ROUTER or REPEATER instead) nodes act as network infrastructure while still showing up actively in the node list and accepting messages as a primary user device.

Use when: You want a node to serve as infrastructure (good location, elevated) but also receive your own messages. Common for operators who have a node at home on a mast and want both routing benefit and personal message delivery to that node.

Note: ROUTER_CLIENT (deprecated; use ROUTER or REPEATER instead) uses somewhat more airtime than pure ROUTER because it participates more actively in NodeInfo and channel activity. If the node is purely infrastructure, use ROUTER.

REPEATER

The most minimal infrastructure role. REPEATER nodes:

- Rebroadcast all received packets immediately (highest rebroadcast priority, even higher than ROUTER)

- Do NOT send NodeInfo broadcasts - they are intentionally invisible in the node list
- Do NOT participate in messaging - they cannot send or receive application-layer messages
- Consume minimal airtime overhead

Use when: Deploying a relay-only node in a gap in coverage where you only need packet forwarding and don't need the node to participate in messaging or appear in the network map. REPEATER nodes are ideal for embedded or remotely deployed relays with no user interaction.

Caution: Because REPEATER nodes don't send NodeInfo, they won't appear on your map or in your node list. This makes it harder to verify they are online. Plan for out-of-band monitoring (serial console, MQTT telemetry, physical LED) if using REPEATER in unattended deployments.

TRACKER

Optimized for asset and vehicle tracking use cases. TRACKER nodes:

- Send frequent position updates automatically
- Are optimized to conserve bandwidth by only broadcasting position data (they reduce other overhead)
- Can be configured with smart beaconing: transmit when moving, reduce rate when stationary

Use when: Attaching a node to a vehicle, pet, person, or asset for location tracking. The TRACKER role signals to the firmware and other nodes that this device's primary purpose is position reporting.

TAK

Designed for interoperability with TAK (Team Awareness Kit) - tactical situational awareness software used by military, law enforcement, SAR teams, and emergency management. TAK role nodes:

- Format outgoing position packets in a way compatible with TAK server ingestion
- Enable TAK-specific extensions in the Meshtastic packet format
- Work alongside ATAK (Android TAK) and WinTAK clients

Use when: Integrating Meshtastic into a TAK-based operational picture. Requires TAK server infrastructure or ATAK-capable devices on the network. Not useful for general community mesh deployments.

SENSOR

Optimized for sensor nodes that transmit telemetry data (temperature, humidity, air quality, power levels, etc.) rather than user messages. SENSOR nodes:

- Prioritize low power consumption over routing participation
- May sleep between sensor readings

- Reduce rebroadcast activity to conserve battery

Use when: Deploying a battery-powered environmental sensor, weather station, or power monitor. The node's primary job is to report readings, not to relay traffic.

LOST_AND_FOUND

A specialized tracker role for lost-item finders (similar in concept to Apple AirTags or Tile trackers but on the Meshtastic mesh). Lost_And_Found nodes:

- Broadcast their position at a very low duty cycle to extend battery life
- Are designed for small form-factor nodes attached to items (bags, bikes, pets)
- Other mesh nodes that receive the beacon can report the position on the finder's behalf

Use when: Building a lost-item tracker on Meshtastic hardware. Not suitable for primary communication or infrastructure roles.

Serial Console Enabled

Config key: `device.serial_enabled`
Default: `true`

Controls whether the firmware exposes a serial console on the USB/UART port. When enabled, you can connect to the node via USB serial at 115200 baud and interact with the device (view logs, run CLI commands). When disabled, the serial port is not active.

Disable when: Deploying a node in an environment where unauthorized USB access is a concern, or to reduce power consumption marginally in deeply embedded deployments. Most users should leave this enabled - it is the primary recovery mechanism if you lose network access to the node.

Warning: If you disable Serial Console and also lose WiFi/Bluetooth access to the node, recovery may require a full firmware reflash.

Debug Log Enabled

Config key: `device.debug_log_enabled`
Default: `false`

When enabled, the firmware outputs verbose debug messages to the serial console. These messages include detailed information about packet processing, routing decisions, radio layer events, and subsystem state changes.

Enable when: Troubleshooting connectivity issues, investigating unexpected behavior, or developing integrations. Debug logging produces a high volume of output that can make normal serial console use harder to read.

Disable in production: Debug logging adds slight CPU overhead and can fill serial buffers. Leave disabled on deployed infrastructure nodes unless actively troubleshooting.

Rebroadcast Mode

Config key: `device.rebroadcast_mode`
Default: `ALL`

Controls which received packets this node will rebroadcast (relay) to other nodes. This setting interacts with the Role setting - Role determines the priority and frequency of rebroadcasting; Rebroadcast Mode determines what gets rebroadcast at all.

ALL

The default mode. This node rebroadcasts all received packets that pass the hop limit and deduplication filters, regardless of source or content. This is the correct mode for most nodes.

ALL_SKIP_DECODING

The node rebroadcasts all packets but skips attempting to decode their payload. This mode is primarily useful for REPEATER-role nodes where you want maximum rebroadcast speed and don't need the node to process message content. Slightly reduces CPU overhead per packet.

Use when: Operating a dedicated relay node where content processing is unnecessary and you want to minimize latency and CPU load.

LOCAL_ONLY

The node only rebroadcasts packets that originate from nodes it can hear directly (one hop away). Packets that have already been relayed by another node (hop count reduced) are not rebroadcast. This effectively limits the node's rebroadcast radius to its immediate RF neighborhood.

Use when: You have a densely packed area of nodes and want to limit the rebroadcast storm that can occur when many nodes all relay the same packet. LOCAL_ONLY can significantly reduce channel utilization in dense deployments.

KNOWN_ONLY

The node only rebroadcasts packets from nodes that appear in its local node database (nodes it has seen NodeInfo from). Unknown nodes - those that have never sent a NodeInfo the local node has received - are not relayed.

Use when: Running a private or semi-private mesh where you want to restrict relay behavior to known network members. This can reduce relay of stray packets from neighboring networks that share the same channel.

Node Info Broadcast Interval

Config key: `device.node_info_broadcast_secs`
Default: Role-dependent (CLIENT: 900 seconds / 15 minutes; ROUTER: 3600 seconds / 1 hour)

How often (in seconds) the node broadcasts a NodeInfo packet - the announcement that includes the node's name, hardware model, and public key. NodeInfo packets allow other nodes to know you exist and update their neighbor lists.

Lower values: More frequent announcements - nodes see each other more quickly after joining the network, and the map updates faster. Costs more airtime.

Higher values: Less frequent announcements - reduces channel utilization overhead, especially important on busy networks. Fixed infrastructure nodes (ROUTER, REPEATER) should use longer intervals (3600 - 10800 seconds) since their presence is stable.

Minimum recommended: 300 seconds (5 minutes) for active portable nodes. Going below this starts to noticeably consume channel airtime with overhead traffic.

Double-Tap as Button

Config key: `device.double_tap_as_button_press`
Default: `false`

On devices with an accelerometer (such as the RAK WisBlock with RAK1904 accelerometer module), enabling this option allows a physical double-tap on the device to simulate a button press. Useful for waking a device from sleep or triggering notifications on nodes that don't have physical buttons.

Enable when: Using a device in a case without easy access to physical buttons, or when you want tap-to-wake functionality. Requires compatible hardware with an accelerometer.

Managed Mode

Config key: `device.is_managed`
Default: `false`

When Managed Mode is enabled, the node's configuration can only be changed via the admin channel - it cannot be modified locally through the app's direct Bluetooth connection or the web interface. This is a security feature for remotely deployed infrastructure nodes.

Enable when: Deploying a ROUTER or REPEATER node that should be centrally managed and protected from unauthorized local modification. For example, a rooftop node that community members can physically access - Managed Mode prevents casual configuration changes.

Warning: Enabling Managed Mode without first configuring an admin channel (see Admin Channel Index below) will lock you out of local configuration. Ensure the admin channel is properly set up before enabling this.

Admin Channel Index

Config key: `device.admin_channel_index`
Default: `0` (primary channel)

Specifies which channel index (0 - 7) is designated as the admin channel for remote configuration. Admin channel messages can change device configuration, reboot the node, and perform other privileged operations - all encrypted with the admin channel's PSK.

Setting this to a non-zero channel index lets you use a private secondary channel (with its own PSK) specifically for administrative messages, keeping admin traffic separate from user messages. For example:

- Channel 0: public mesh channel (LongFast or community key)
- Channel 1: private admin channel (strong PSK known only to node operators)
- Admin Channel Index: 1

This configuration means only operators with the channel 1 PSK can remotely reconfigure the node, while anyone on channel 0 can use the mesh normally.

Best practice for infrastructure nodes: Always use a dedicated admin channel with a strong PSK and set Admin Channel Index accordingly. Never leave admin access on the default public channel in a production deployment.

Position Configuration Settings

The Position configuration section (**Config > Position**) controls how your node acquires, reports, and manages GPS and location data. Getting position configuration right affects mesh map accuracy, battery life, and channel airtime - particularly important for mobile nodes and large networks.

Access these settings in the Meshtastic app under **Settings > Radio Configuration > Position**, or via the Python CLI with `meshtastic --set position.*`.

GPS Mode

Config key: `position.gps_mode`
Default: `ENABLED` (on devices with GPS hardware)

Controls the GPS receiver's operating state. This is a top-level switch that determines whether GPS hardware is used at all.

ENABLED

The GPS receiver is active and continuously seeks satellite fixes. The node uses live GPS data for position reporting. This is the default for devices with GPS hardware (e.g., T-Beam, WisBlock with RAK1910/RAK12500 GPS module).

DISABLED

GPS is disabled. The node will use a fixed position if one has been set (see Fixed Position below), or will report no position if no fixed position is configured. Disabling GPS when you don't need real-time position significantly reduces power consumption.

Use when: The node is permanently fixed and its position is set manually via Fixed Position. A rooftop ROUTER node, for example, has no reason to run its GPS continuously - set a fixed position once and disable GPS to save power.

NOT_PRESENT

Informs the firmware that no GPS hardware is present on this device. This prevents the firmware from attempting to initialize GPS hardware that doesn't exist, which can cause startup delays and

error log noise. Set this on devices without any GPS module (e.g., a bare ESP32 LoRa board without GPS, or a RAK4631 with no GPS WisBlock attached).

Fixed Position

Config key: Set via app "Set Fixed Position" action or CLI `meshtastic --setlat <lat> --setlon <lon> --setalt <alt>`
Default: Not set

A manually entered GPS coordinate that the node broadcasts as its position instead of (or in addition to, depending on GPS Mode) GPS-derived coordinates. Fixed position is stored in non-volatile memory and persists across reboots.

Use cases:

- Fixed infrastructure nodes (rooftop ROUTER, indoor gateway) - set their exact location once, disable GPS
- Devices without GPS hardware - allow them to appear on the map at a known location
- Privacy-conscious users who want to broadcast an approximate location rather than precise GPS coordinates

Setting a fixed position:

- In the Meshtastic app: Settings > Radio Configuration > Position > "Set to current location" or enter coordinates manually
- Via CLI: `meshtastic --setlat 37.7749 --setlon -122.4194 --setalt 15`
- Via Python API: `iface.localNode.setPosition(lat=37.7749, lon=-122.4194, alt=15)`

To clear a fixed position: Use `meshtastic --remove-position` in the CLI.

Position Broadcast SMART Enabled

Config key: `position.position_broadcast_smart_enabled`
Default: `true`

Smart position broadcasting adapts broadcast frequency based on movement. When enabled, the node transmits position updates more frequently when moving and less frequently when stationary. This significantly reduces channel airtime for mobile nodes compared to fixed-interval broadcasting.

How smart beaconing works:

1. If the node has moved more than the configured Minimum Distance since the last broadcast, it broadcasts immediately (regardless of elapsed time)
2. If the node has not moved, it waits up to the configured Broadcast Interval before broadcasting
3. Speed is factored in: faster movement triggers more frequent updates

Enable for: Any mobile node (vehicle tracker, handheld carried by a walking/driving user). Smart beaconing is almost always beneficial for mobile nodes.

Consider disabling for: Fixed infrastructure nodes with a set position - they should broadcast at a fixed, low-frequency interval (see Broadcast SECS below) rather than smart beaconing, which adds minor computational overhead.

Broadcast SECS (Position Broadcast Interval)

Config key: `position.position_broadcast_secs`
Default: `900` (15 minutes)

The maximum interval in seconds between position broadcasts. When Smart Beaconing is enabled, this is the upper bound - the node will not go longer than this interval without broadcasting, even if stationary. When Smart Beaconing is disabled, this is the fixed broadcast interval.

Guidance by use case:

Use Case	Recommended Interval	Rationale
Vehicle tracker (active event)	60 - 120 seconds	Frequent updates needed for real-time tracking
Hiking/walking node	120 - 300 seconds	Balance between track fidelity and airtime
Fixed CLIENT node	900 - 1800 seconds	Position doesn't change; reduce overhead
Fixed ROUTER/infrastructure	3600 - 10800 seconds	Minimal overhead for stable fixed nodes

Channel utilization impact: Position packets are among the longer Meshtastic packets. On a busy network with many nodes, unnecessarily frequent position broadcasts are a significant source of channel congestion. Always use the longest interval consistent with your tracking needs.

Smart Minimum Distance

Config key: `position.broadcast_smart_minimum_distance`
Default: `100` meters

When Smart Position Broadcasting is enabled, this is the minimum distance (in meters) the node must travel from its last broadcast location before a new position broadcast is triggered by movement. If the node moves less than this distance, the movement does not by itself trigger a new broadcast.

Tuning guidance:

- **Walking/hiking:** 50 - 100 meters - captures meaningful position changes at walking speed
- **Vehicle tracking:** 100 - 500 meters - avoids rapid-fire updates at slow speeds (parking lots, traffic)
- **Emergency/rescue:** 25 - 50 meters - fine-grained position updates for close-range coordination

---

GPS Update Interval

Config key: `position.gps_update_interval`
Default: `120` seconds (2 minutes)

How often (in seconds) the firmware polls the GPS module for a new position fix. This is distinct from the broadcast interval - the GPS may update frequently internally while position broadcasts happen less often.

A shorter GPS update interval means the firmware always has a more current fix available when it decides to broadcast. A longer interval reduces GPS power consumption (the GPS receiver is one of the most power-hungry components on nodes like the T-Beam).

Recommended values:

- Active mobile use: 30 - 60 seconds
- Occasional position checks: 120 - 300 seconds
- Mostly stationary: 600 - 1800 seconds (or disable GPS and use fixed position)

Position Flags

Config key: `position.position_flags`
Default: Altitude + Speed + Heading (varies by firmware version)

Position flags are a bitmask that controls which optional data fields are included in position packets. Each additional field adds bytes to the packet, increasing airtime. Choose only the flags relevant to your use case.

Flag	Data Added	Packet Size Impact	Use When
ALTITUDE	Elevation in meters MSL	+4 bytes	Mountainous terrain, aviation, 3D positioning
ALTITUDE_MSL	Altitude above mean sea level (vs ellipsoid)	+4 bytes	When sea-level altitude is specifically needed
GEOIDAL_SEPARATION	Difference between WGS84 ellipsoid and geoid	+4 bytes	Precision surveying; not needed for most uses
DOP	Dilution of Precision (accuracy estimate)	+2 bytes	When position accuracy qualification is important
HVDOP	Horizontal and Vertical DOP separately	+4 bytes	Precision tracking applications
SATINVIEW	Number of GPS satellites in view	+1 byte	Diagnostics, signal quality assessment
SEQ_NO	Sequence number for packet ordering	+2 bytes	When multiple position sources must be ordered
TIMESTAMP	Unix timestamp of GPS fix	+4 bytes	When time-of-fix (vs time-of-transmission) matters
HEADING	Course over ground in degrees	+2 bytes	Vehicle and vessel tracking, direction of travel
SPEED	Speed over ground in m/s	+2 bytes	Vehicle tracking, activity analysis

Minimal position packet (lat/lon only): Omit all optional flags. Suitable for fixed nodes or simple presence-only tracking.

Full vehicle tracking: Enable Altitude, DOP, Heading, Speed. Gives a complete picture without the rarely useful fields.

Airtime consciousness: On a 250 kbps data rate channel, each position packet with all flags enabled may be 40+ bytes longer than a minimal packet, translating to measurably more on-air time per broadcast.

GPS Attempt Time

Config key: `position.gps_attempt_time`
Default: `900` seconds (15 minutes)

The maximum time the firmware will attempt to acquire a GPS fix before giving up and going to sleep (in power-save GPS mode). If the GPS acquires a fix before this timeout, it sleeps early. If it cannot get a fix within this window, it stops trying until the next GPS update cycle.

In open sky environments: A fix is typically acquired within 30 - 60 seconds. A 900-second attempt time is very generous - most GPS activity will complete long before timeout.

In challenging environments (indoors, urban canyons, dense tree cover): GPS may struggle to get a fix. A longer attempt time gives more opportunity for a fix; a shorter time saves power by giving up sooner when conditions make a fix unlikely.

GPS Power Management (Power Saving)

Config key: `position.gps_power_mode` (or handled automatically based on GPS Mode and Update Interval)
Default: Automatic

On battery-powered nodes, GPS is one of the largest power consumers (typically 20 - 50 mA when active, vs 2 - 5 mA for the LoRa transceiver in sleep). Several strategies minimize GPS power draw:

Duty-Cycle GPS (Recommended for Mobile Nodes)

The GPS receiver powers on only when a new fix is needed (based on GPS Update Interval), acquires a fix, then powers down. Between updates, the GPS is completely off. This can reduce GPS-related power consumption by 90%+ compared to continuous operation.

Meshtastic handles this automatically when GPS Mode is ENABLED and GPS Update Interval is set to a value greater than the GPS Attempt Time.

Fixed Position + GPS Disabled (Best for Fixed Nodes)

The most power-efficient approach for fixed nodes: enter the position manually once, set GPS Mode to DISABLED. The GPS receiver is never powered, eliminating all GPS power consumption entirely.

GPS NOT_PRESENT (For Devices Without GPS)

On boards without GPS hardware, setting GPS Mode to NOT_PRESENT prevents any attempt to initialize GPS, eliminating initialization delay and preventing power being applied to a non-existent module.

Practical Power Impact

GPS Mode	Approximate Current Draw	Use When
Continuous (always on)	30 - 50 mA continuous	Real-time tracking where every second matters
Duty-cycle (120s interval)	2 - 8 mA average	Standard mobile node
Duty-cycle (600s interval)	0.5 - 2 mA average	Low-power mobile node
DISABLED (fixed position)	0 mA for GPS	Fixed infrastructure nodes

Network Configuration Settings

The Network configuration section (**Config > Network**) controls how ESP32-based Meshtastic nodes connect to IP networks - WiFi and Ethernet - and associated services like NTP and remote logging. These settings are only relevant for ESP32 hardware; nRF52-based boards (like the RAK4631) do not support WiFi/Ethernet and these settings have no effect on them.

Access these settings in the Meshtastic app under **Settings > Radio Configuration > Network**, or via the Python CLI with `meshtastic --set network.*`.

Note: Network connectivity unlocks important Meshtastic features: the web interface, MQTT gateway, APRS bridging, NTP time synchronization, and remote syslog. If you are using ESP32-based hardware (T-Beam, T-Lora, Heltec WiFi LoRa, Station G2, etc.), understanding these settings is important for gateway and infrastructure deployments.

WiFi SSID

Config key: `network.wifi_ssid`
Default: Empty (WiFi disabled)

The SSID (network name) of the WiFi network the node should connect to as a client. Case-sensitive. Maximum 32 characters.

To enable WiFi: Set both WiFi SSID and WiFi Password. The node will attempt to join the network on boot and reconnect if the connection drops.

Important considerations:

- WiFi and Bluetooth can operate simultaneously on most ESP32 devices, but enabling WiFi increases power consumption significantly (typical WiFi active current: 80 - 200 mA vs 5 - 10 mA for LoRa+BT only)
- A node with WiFi enabled and a good internet connection can serve as an MQTT gateway, APRS-IS gateway, and NTP time source for the mesh
- On battery-powered mobile nodes, WiFi should generally be disabled to preserve battery life. Enable WiFi on mains-powered infrastructure nodes.

WiFi Password

Config key: `network.wifi_psk`
Default: Empty

The WPA2 password for the WiFi network specified by WiFi SSID. Stored in the device's flash memory. Supports open networks (leave password empty if the network has no password, though this is strongly discouraged for security reasons).

Security note: The WiFi password is stored in plaintext in the device's NVS (Non-Volatile Storage) flash partition. Anyone with physical access to the device and a serial connection can potentially extract it. Do not configure a node with your primary home WiFi password on a device that could be physically compromised; consider using a dedicated IoT VLAN or guest network.

WiFi Mode

Config key: `network.wifi_mode` (indirectly controlled via presence of SSID and AP settings)
Default: Client mode when SSID is set

ESP32-based Meshtastic nodes support three WiFi operating modes:

Client Mode (STA)

The node connects to an existing WiFi network as a client (station). This is the standard mode for gateway nodes that need internet access. In client mode, the node:

- Obtains an IP address via DHCP (or a configured static IP)
- Can reach the internet for MQTT, NTP, APRS-IS, and other services
- Is accessible on the local network at its assigned IP address
- Serves the web interface at `http://<node-ip>/`

Access Point Mode (AP)

The node creates its own WiFi access point instead of connecting to an existing network. Other devices (phones, computers) connect directly to the node's WiFi AP to access the web interface, without needing an external router or internet connection.

AP mode is configured by: Leaving WiFi SSID empty (or setting it to the AP name) and enabling the AP in the app's Network settings. The default AP SSID is typically `meshtastic-XXXX` (where XXXX is derived from the device MAC address). Default AP password: `12345678`.

Use AP mode when:

- In the field with no known WiFi network - AP mode lets you connect your phone directly to the node for configuration without needing Bluetooth
- Setting up a node in an environment where you don't control the WiFi infrastructure
- Demonstrating Meshtastic at an event where local WiFi is unavailable or unreliable

AP mode limitations: No internet access for the node (no MQTT, NTP, APRS), and the phone connected to the AP loses its normal internet access while connected. Not suitable for gateway deployments.

AP + Client Mode (Soft AP + STA)

The node simultaneously connects to an existing WiFi network (client mode) AND creates its own access point. This allows devices to connect directly to the node's AP while the node itself maintains internet connectivity through the upstream network.

Use when: You want both - internet-connected gateway functionality AND the ability to connect phones/laptops directly via WiFi without knowing the upstream network password. Common for demo setups and ARES deployments where field operators need web interface access.

Note: Running both modes simultaneously increases power consumption and can reduce WiFi throughput due to the ESP32 sharing radio time between AP and STA roles.

Ethernet Enabled

Config key: `network.eth_enabled`
Default: `false`

Enables the Ethernet interface on hardware that supports it. Currently, the primary supported Ethernet-capable Meshtastic hardware is the **Meshtastic Station G2** (which uses a W5500 SPI Ethernet module) and some custom RAK WisBlock builds with Ethernet modules.

Advantages of Ethernet over WiFi for infrastructure nodes:

- More reliable and stable connection - no RF interference, no re-association delays
- Lower and more consistent latency
- Lower power consumption than WiFi (typically 10 - 20 mA vs 80 - 200 mA for WiFi active)
- No PSK to manage or expose

Use when: Deploying a fixed infrastructure node (ROUTER or gateway) at a location with Ethernet infrastructure - server room, communications closet, network rack. Ethernet-connected gateway nodes are more reliable than WiFi-connected ones for long-term unattended operation.

NTP Server

Config key: `network.ntp_server`
Default: `0.pool.ntp.org`

The hostname or IP address of the NTP (Network Time Protocol) server the node uses to synchronize its real-time clock. Accurate time is important for:

- Correct message timestamps displayed in the app
- Log entries with accurate timestamps for troubleshooting
- MQTT message timestamps
- Coordinating time-dependent operations across the mesh

Meshtastic nodes without internet connectivity rely on time received from other nodes on the mesh (nodes share timestamps in packets). A node with NTP access becomes a time source for the entire mesh, improving timestamp accuracy for all nodes.

Recommended NTP Servers

Server	Description	Use When
`0.pool.ntp.org`	Default NTP pool (global)	General use, internet-connected nodes
`time.cloudflare.com`	Cloudflare NTP (anycast, fast)	Reliable alternative with good global coverage
`time.google.com`	Google Public NTP	Reliable alternative
`time.nist.gov`	NIST time server	When US government standard time is needed
`192.168.x.x` (local)	Your own local NTP server	Isolated networks, high-accuracy requirements, no internet

Local NTP server: If your deployment has a local NTP server (common in enterprise and government networks, and in some emergency operations centers), set this to that server's address. This reduces internet dependency and may improve synchronization accuracy.

NTP without internet: If the node has no internet access but is on a local network with a router that provides NTP (most home routers do), using the router's IP address as the NTP server works well: `192.168.1.1` or similar.

---

rsyslog Server

Config key: `network.rsyslog_server`
Default: Empty (disabled)

Configures remote syslog logging. When set to a hostname or IP address (with optional port, e.g., `192.168.1.100:514`), the node sends its log output to a remote syslog server over UDP using the standard syslog protocol (RFC 3164/5424).

Why use remote logging:

- Centralized log collection from multiple nodes - see all infrastructure logs in one place
- Persistent log storage - node logs that would otherwise be lost on reboot are captured on the server
- Real-time alerting - syslog servers can trigger alerts on specific log messages (errors, reconnections, etc.)
- Troubleshooting unattended nodes - diagnose issues without physically connecting a serial cable

Setup requirements:

- A syslog server running on the local network (rsyslog, syslog-ng, Graylog, or any standard syslog receiver)
- The node and syslog server must be on the same network (or routable to each other)
- UDP port 514 must be accessible (default syslog port)

Recommended syslog servers for small deployments:

- **rsyslog** on Linux (Raspberry Pi, server): Simple, lightweight, standard
- **Graylog**: Full log management with search and dashboards - good for larger deployments
- **Loki + Grafana**: Modern log aggregation with excellent visualization

Example rsyslog configuration to receive Meshtastic logs on a Linux server:

```
# /etc/rsyslog.d/meshtastic.conf
module(load="imudp")
input(type="imudp" port="514")

# Save Meshtastic logs to a dedicated file
if $fromhost-ip == '192.168.1.50' then /var/log/meshtastic/node1.log
```

IPv6

Config key: `network.ipv6_enabled` (handled automatically in current firmware)
Default: Enabled when available

Meshtastic's ESP32 networking stack (based on ESP-IDF and lwIP) supports IPv6. The node will request an IPv6 address via SLAAC (Stateless Address Autoconfiguration) if the connected network provides IPv6 router advertisements.

For most users, IPv6 is transparent - the node simply has both an IPv4 and IPv6 address, and connections work over whichever is available. No explicit configuration is typically needed.

When IPv6 matters:

- Networks that are IPv6-only (rare but increasingly common in enterprise environments)
- When you want to access the node's web interface over IPv6 (useful on networks where IPv4 DHCP is restricted)
- Future MQTT and NTP configurations that specify IPv6 server addresses

---

Practical Configuration Guidance

Standard Home Gateway Node

For a mains-powered T-Beam or Station G2 acting as a gateway and router at home:

- WiFi SSID: your home network SSID (or use IoT VLAN if available)
- WiFi Password: your network password
- WiFi Mode: Client
- NTP Server: `0.pool.ntp.org` (default is fine)
- rsyslog Server: empty unless you have a log server

Field Deployment - No Internet

For a node deployed at an event or emergency operation without internet access:

- WiFi Mode: AP (create your own access point for phone/laptop connection)
- AP SSID: something recognizable, e.g., "ARES-Mesh-Node1"
- NTP Server: empty or your EOC's local network NTP if available
- Time will sync from other mesh nodes that have NTP access

Ethernet-Connected Infrastructure

For a fixed ROUTER node on a rack or network closet:

- Ethernet Enabled: true
- WiFi: disabled (leave SSID empty)

- NTP Server: your organization's NTP server or `time.cloudflare.com`
- rsyslog Server: your organization's syslog collector
- Managed Mode: true (with admin channel configured)

Direct Phone Connection Without Bluetooth

When Bluetooth is unavailable or problematic (interference, pairing issues):

- WiFi Mode: AP (or AP+Client if the node also needs internet)
- Connect your phone to the node's AP network
- Open `http://meshtastic.local` in a browser or use the Meshtastic app with HTTP transport
- Full configuration and messaging capability without Bluetooth