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.

Revision #7
Created 2026-05-03 05:50:05 UTC by Mesh America Admin
Updated 2026-06-09 00:22:45 UTC by Mesh America Admin