White Paper: Using *rxdelay*, *txdelay* and *direct.txdelay* to Minimize Collisions in a MeshCore Routing

Abstract

This paper examines the purpose of a repeater's *rxdelay*, *txdelay* and *direct.txdelay* parameters and their influence on collision probability within the MeshCore's routing engine.

txdelay governs randomized retransmission timing for **flood-based** propagation, while ***direct.txdelay*** applies similar timing randomization to **direct/unicast** traffic. Both use the canonical backoff equation:

$$t=[5 \times \text{airtime} \times \text{“}txdelay\text{”}]$$

rxdelay has a similar random effect but only on reception.

We graph the timing windows to show how the *txdelay* and *direct.txdelay* parameters reduce simultaneous transmissions to reduce collisions and the retransmission of messages on a MeshCore mesh.

We propose a process to manually set repeater *rxdelay, txdelay* and *direct.txdelay* based on its local environment, using a count of the number of active/good signal strength neighbors within a single hop (0-hop, Direct) of a repeater.

1. Introduction

Flood-based mesh networks rely on randomized retransmission timing to prevent synchronized forwarding events. MeshCore exposes two timing parameters:

- *txdelay* — controls randomization for **flood routing**
- *direct.txdelay* — controls randomization for **direct/unicast traffic**

Both parameters reduce collision probability, but they operate in different traffic domains and must be tuned with different priorities.

2. Collision Dynamics in Mesh Networks

When multiple nodes receive or generate messages at nearly the same moment, they tend to transmit simultaneously unless randomized. Collisions occur when overlapping transmissions exceed the receiver's capture threshold preventing reception of the message, resulting in it not being heard or received. MeshCore mitigates this through timing randomization applied separately to flood and direct transmissions and reception.

3. MeshCore Backoff Equations

MeshCore Source Code

MeshCore/examples/simple_repeater/MyMesh.cpp [version 1.13.0]

```
// defaults
_prefs.airtime_factor = 1.0;          // one half
_prefs.rx_delay_base = 0.0f;          // turn off by default, was 10.0;
_prefs.tx_delay_factor = 0.5f;        // was 0.25f
_prefs.direct_tx_delay_factor = 0.2f; // was zero
```

//equations

// The txdelay and direct.txdelay calculations for backoff are "identical" (same equations) but apply to different types of messages

```
uint32_t  MyMesh::getRetransmitDelay(const mesh::Packet *packet) {
uint32_t  t = (_radio->getEstAirtimeFor(packet->path_len + packet->payload_len + 2) *
                                                        _prefs.tx_delay_factor);
return getRNG()->nextInt(0, 5*t + 1);
}
```

```
return (int)((pow(_prefs.rx_delay_base, 0.85f - score) - 1.0) * air_time);
```

The default value of *txdelay* is equal to **0.5** and the default value of *direct.txdelay* is equal to **0.2** (this is displayed as .1999999, not .2 since it is a float that is quantized for the displayed value)

The default value of r*xdelay* is equal to **0**

The Retransmit Delay (backoff) function in MeshCore is done in firmware and not done in hardware by the radio controller device. As *txdelay* is increased, the number of random "slots" increases, which decreases the probability of a collision (they are transmitted in different slots). A "slot" time is the calculated time of the transmission of the current message that's ready to send.

It is important that these values be configured in online repeaters via a CLI command to something other than the default values (ex CLI commands: set *rxdelay* 3, set *txdelay* 1.1 and set *direct.txdelay* .5).

- The default *direct.txdelay* sets message transmission to **immediate** (no delay). This means there is a “guaranteed” collision.
- The default *rxdelay* sets message reception delay to **immediate** (no delay). This can increase the probability of a collision.
- The default *txdelay* sets transmission to one of three transmission slots which has a 12.5% probability of collision if two nearby repeaters receive a Flood message and both determine it needs to be transmitted. This is a high probability of a collision, which increases with additional neighbors retransmitting a Flood or Ack message.

Throughout the rest of this paper, when the term *txdelay* is used, it generally applies to both *txdelay* and *direct.txdelay* unless *direct.txdelay* is specifically stated.

3.1 Flood Backoff (*txdelay*)

Flood retransmission timing uses:

$$t = \lfloor 5 \times \text{airtime} \times txdelay \rfloor$$

The retransmission delay is:

$$\text{getRNG()->nextInt(0, 5*}t\text{ + 1);}$$

t defines the number of airtime-sized slots available for randomization.

This equation generates a random floating-point number and then converts it to an Integer (a counting number, no fractions) from between the lower bound (0) and upper bound (0, 5*t + 1) which gives a uniform distribution within a specified range. Every value between the lower bound and upper bound has an equal probability of being selected.

3.2 Interpretation of the 5× Multiplier

With a normalized airtime (default setting = 1.0):

$$t = \lfloor 5 \times txdelay \rfloor$$

This creates discrete threshold "steps" (see Figure 2) where small changes in *txdelay* do not always create an additional transmit slot to reduce the probability of a collision.

3.3 Collision Probability as a Function of *t*

Collision probability decreases approximately with:

$$P_{\text{collision}} \propto \frac{1}{t}$$

Caveats regarding the following figures:

In the two graphs below *txdelay* (and implicitly *direct.txdelay*) were arbitrarily assigned as:
Home repeater 0.3 to 0.7 (in steps of 0.1)
Local repeater 0.7 to 1.4 (in steps of 0.1)
Regional repeater 1.4 to 3.0 (in steps of 0.1)

This is not the proposed grouping; it was simply used to generate a segmented and colored graph. Please note that *txdelay* cannot be set above 2.0 via the CLI. The graph goes to 3.0 to show the trend.

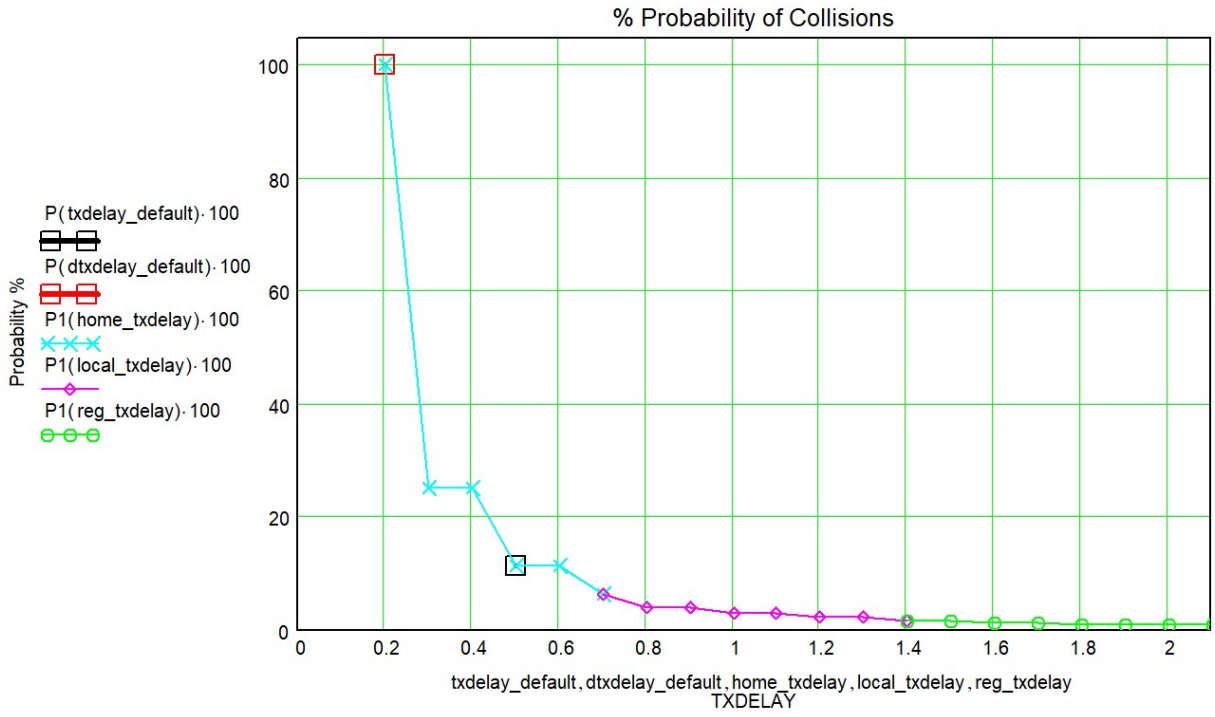
The following chart demonstrates the reduction in the % Probability of a Collision (with another device set at the same *txdelay* value) as *txdelay* increases.

Please Note:
The default *direct.txdelay* value has a 100% probability of a collision.
The default *txdelay* value has a 12.5% probability of a collision.
A difference of 0.2 in the *txdelay* value is required to see an additional slot. (Small differences have no real effect).

This only shows the probability of two repeaters choosing the same slot (i.e. both pick slot 1). But the actual probability is higher, for example if both are choosing from 5 slots, there are actually 5 possible collisions (both pick 1, both pick 2, etc). So, the probability is 5 times higher than shown in the figure (n times higher for n Slots). Plus, with multiple repeaters the probability of collision goes up since they all could be choosing the same slot. The purpose of these figures is to show the "trend" (steep drop and then tapering off) of the probability with increasing *txdelay*.

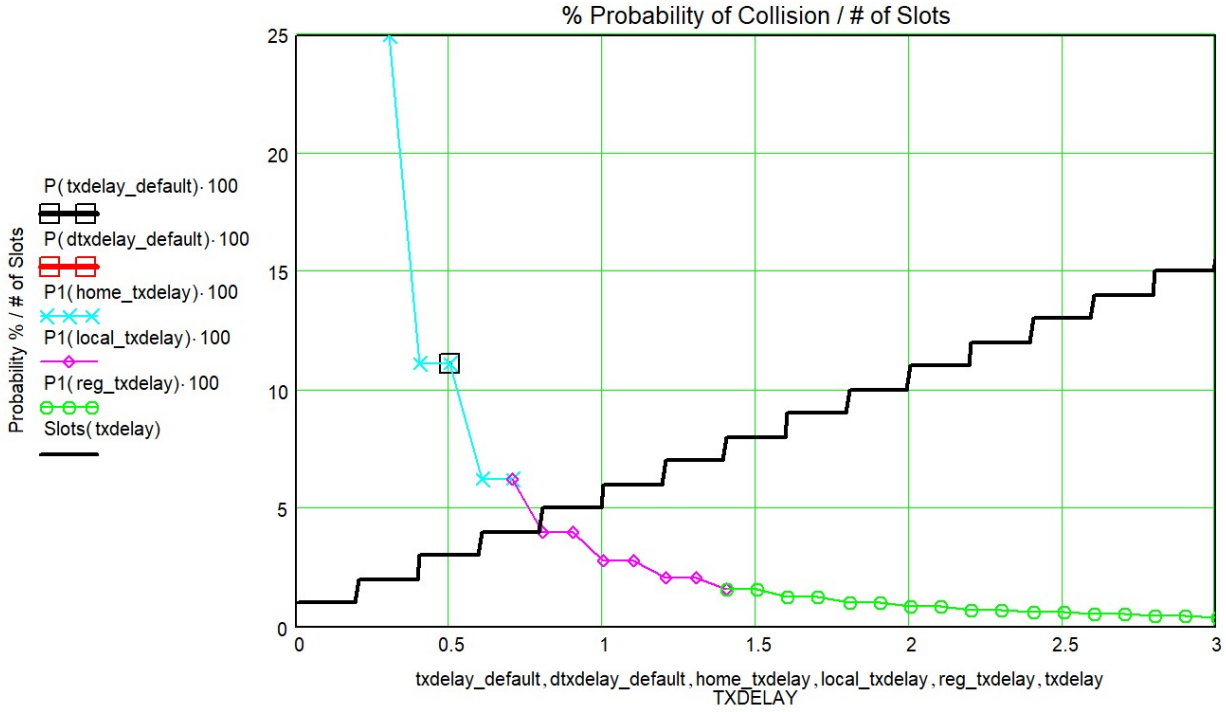
This is a major reason the number of neighbors is a key factor in setting the *txdelay* value. For example: if you have twice as many neighbors repeating the same message, the probability of a collision ~ doubles. This effect is not shown in the figures.

Figure 1.



The following chart is a “Zoomed in” version of figure 1 and shows the increase in the # of Slots as *txdelay* increases.

Figure 2.



Please Note:
The % probability does decline beyond 2 as *txdelay* is increased but doesn’t have a significant effect. However, increasing *txdelay* will delay the transmission of a message without a significant decrease in the probability of a collision. Values greater than ~2.5 provide no benefit.

3.4 Flood Timing Threshold Examples

3.4.1 Timing Slot Expansion as *txdelay* Increases

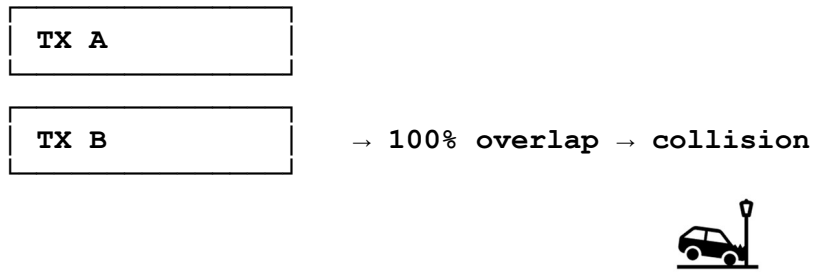
t = floor(5 × airtime × *txdelay*)
airtime = 1.0

txdelay:	0.2	0.5	1.0	2.0
5×:	1.0	2.5	5.0	10.0
t:	1	2	5	10

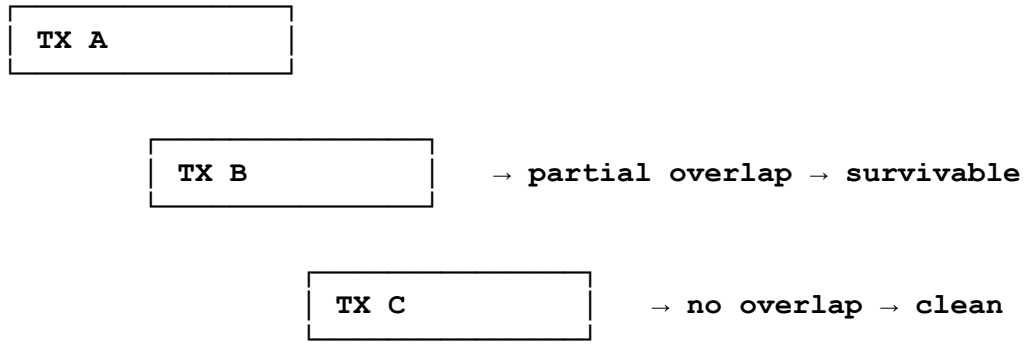
Slots:

t=1	t=2	t=5	t=10
[0]	[0] [1]	[0] [1] [2] [3] [4]	[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

Without Randomization (t = 0)



With Randomization (t = 6)



3.4.2 Collision Behavior with Increasing *txdelay*

txdelay	5×*txdelay*	t	Collision Behavior
0.1	0.5	0	Nearly guaranteed collisions
0.5	2.5	2	High collision rate
1.0	5.0	5	Stable in sparse meshes
1.3	6.5	6	Stable in medium meshes
1.9	9.5	9	Very stable
2.0	10.0	10	Extremely stable

3.5 Direct Message Backoff (*direct.txdelay*)

While *txdelay* governs **flood retransmissions**, MeshCore applies a parallel but **narrower** randomization window to **direct/unicast** traffic via *direct.txdelay*.

3.5.1 Conceptual Role

- Reduces collisions when multiple nodes send direct messages at the same moment
- Reduces collisions in room-server, control, and interactive traffic
- Preserves responsiveness by keeping the window smaller than flood timing

- A benefit of DM traffic is that they are generally sent with a path and do not consume the mesh like a flood message (the airtime consumed is considerably less. But they do revert to a Flood when an Ack is not received.

3.5.2 Typical Pairing

Flood (*txdelay*)	Direct (*direct.txdelay*)	Notes
1.0–1.2	0.4–0.6	Sparse meshes
1.3–1.6	0.5–0.8	Medium density
1.8–2.1	0.6–0.9	Dense / hilltop
2.0–2.5	0.6–0.9	Regional

3.5.3 Why direct timing is smaller

- Direct traffic is latency-sensitive
- Direct transmissions involve fewer simultaneous senders (only 1 hop away)
- Flood traffic requires much wider desynchronization

3.6 Direct Message Forwarding and Collision Characteristics

Direct (unicast) messages behave fundamentally differently from flood messages in MeshCore's routing engine. While a flood message is intentionally forwarded by *every* eligible repeater, a direct message is forwarded by **exactly one** repeater per hop — the selected next-hop in the routing path.

3.6.1 Single-Forwarder Property

When a direct message is transmitted:

- Multiple repeaters may **receive** it.
- Only the **designated next-hop** (as determined by the routing engine) will **retransmit** it.
- All other receivers will **discard** the message after deduplication.

This "single-forwarder" behavior means that **a direct message does not fan out**, (goes from one to more than one) and therefore does **not** create the synchronized multi-node retransmission wavefront that flood messages do.

3.6.2 Implication for Collision Probability

Because only one repeater retransmits a given direct message:

- The probability of collision for that message's retransmission is **independent of** *direct.txdelay*.
- There is no competition between multiple nodes forwarding the *same* direct message.
- The backoff window for direct traffic does **not** influence whether that specific message collides with another copy of itself — because no such copies exist.
- If a collision does occur, that message is lost and an Ack is not sent and is not received by the sender. The retry mechanism of the sender will re-transmit the message and hopefully will not collide again. If too many collisions occur messages cannot not get through to the receiver or if they do, the Flood Ack cannot return, and the sender will see a message Fail.

3.6.3 Where *direct.txdelay does* matter

direct.txdelay is not designed to mitigate collisions between copies of the *same* message. Instead, it reduces contention between **independent direct transmissions**, such as:

- Multiple nodes sending direct messages at similar times
- A single node with several direct messages queued close together
- Bidirectional or interactive traffic where both ends transmit concurrently

In these cases, randomized spacing helps prevent overlapping transmissions from unrelated direct flows.

3.7 Temporal Separation Through Tiered *txdelay* Values

MeshCore deployments can assign different *txdelay* values to repeaters based on their functional role or geographic reach. This creates **tiered retransmission windows** that naturally separates ***for example:*** local and regional transmissions in time, reducing cross-tier collision probability.

A local repeater configured with *txdelay* = 1 therefore selects a retransmission delay from:

$$d_{\text{local}} \sim \text{Uniform}(0,5)$$

A regional repeater configured with *txdelay* = 2 selects from:

$$d_{\text{regional}} \sim \text{Uniform}(0,10)$$

3.7.1 Temporal Band Separation

Because the regional node's window is strictly larger, the interval $[5,10]$is **exclusive** to regional retransmissions. Any regional delay drawn from this upper band is **guaranteed non-overlapping** with all possible local retransmissions, which are confined to $[0,5]$.

This creates a natural **temporal hierarchy**:

- **Local repeaters** transmit earlier, occupying the lower portion of the backoff window.
- **Regional repeaters** transmit later, with access to a collision-free tail region unavailable to local nodes.

3.7.2 Operational Benefits

This tiered structure provides several advantages:

- **Reduced cross-tier collisions:** Regional retransmissions occurring in the upper half of their window cannot overlap with local retransmissions.
- **Improved reliability for long-range propagation:** Regional nodes act as a "second wave" of retransmission, reinforcing coverage after local nodes have already attempted delivery.
- **Implicit prioritization:** Local nodes provide fast, low-latency coverage; regional nodes provide slower but more robust reinforcement

3.7.3 Limitations

The guarantee applies only to **cross-tier** interactions:

- Two regional nodes may still collide if both select delays within the same sub-interval.
- Two local nodes may collide if their delays differ by less than one airtime unit.
- The guarantee does not extend to unrelated direct traffic.

Nevertheless, tiered *txdelay* values provide a simple and effective mechanism for **temporal separation** between repeater classes, improving overall flood stability without requiring explicit coordination.

3.8 Receive-Window Randomization (*rxdelay*)

rxdelay introduces controlled entropy into the timing of receive-window openings to prevent deterministic synchronization across nodes. While ***txdelay*** is the primary mechanism for reducing transmit-side collisions, *rxdelay* provides secondary stabilization that prevents timing lockstep, ACK clustering, and hop-to-hop synchronization effects that *txdelay* alone cannot resolve.

3.8.1 Functional Role

After a node transmits or forwards a message, it applies a randomized delay before opening its receive window:

$$t_{\text{rx}} \sim U(0,\text{rxdelay})$$

This offsets the RX window relative to other nodes that heard the same transmission, ensuring that receive-window openings are statistically distributed rather than synchronized.

3.8.2 Interaction with *txdelay*

txdelay and *rxdelay* address different collision modes:

- ***txdelay*** randomizes when nodes transmit, directly reducing simultaneous airtime occupancy.
- ***rxdelay*** randomizes when nodes listen, preventing synchronized receive windows, ACK storms, and deterministic hop-timing patterns.

Because collisions occur during transmission, *txdelay* has the dominant effect on collision probability. *rxdelay* prevents the mesh from falling into repeating timing patterns that *txdelay* alone cannot break.

3.8.3 Collision-Reduction Effect

The probability that two nodes open overlapping RX windows is approximately:

$$P(\text{overlap}) = \frac{airtime}{rxdelay}$$

Once *rxdelay* exceeds roughly 3–4× the message airtime, the marginal reduction in overlap probability becomes small. This is why *rxdelay* does not need to be large to be effective.

rxdelay	Approx overlap probability	Interpretation
2	50%	High overlap; only small meshes tolerate this
3	33%	Good for medium meshes
4	25%	Strong desynchronization; good general setting
5	20%	Dense mesh protection
6	17%	High-fan-out / hilltop stability

3.8.4 Topology-Specific Recommendations

The following values assume typical LoRa airtime for MeshCore deployments and scale proportionally with modulation settings.

Sparse Networks (0–3 nodes, low contention)

- **rxdelay:** 2–3 × airtime
- **Rationale:** Collisions are rare; rxdelay only needs to prevent occasional RX-window alignment.
- **Notes:** Latency should remain minimal; larger rxdelay provides no benefit.

Medium Networks (4–8 nodes, moderate contention)

- **rxdelay:** 3–4 × airtime
- **Rationale:** This range provides sufficient desynchronization to prevent ACK overlap and hop-timing lockstep.
- **Notes:** This is the “default safe” operating region for most deployments.

Dense Networks (9–10 nodes, high mutual visibility)

- **rxdelay:** 4–6 × airtime
- **Rationale:** Dense meshes experience more ACK clustering and receive-window overlap.
- **Notes:** rxdelay above 6× airtime yields diminishing returns unless the topology is highly symmetric.

Regional / Hilltop / High-Fan-Out Networks (repeaters hearing 11–12+ nodes)

- **rxdelay:** 6–8 × airtime

- **Rationale:** Hilltop repeaters receive bursts of messages from many nodes; RX-window desynchronization is critical to avoid ACK storms.
- **Notes:** Larger rxdelay increases latency but significantly improves stability in high-fan-out topologies.

3.8.5 Listen Before Talk (LBT) Backoff

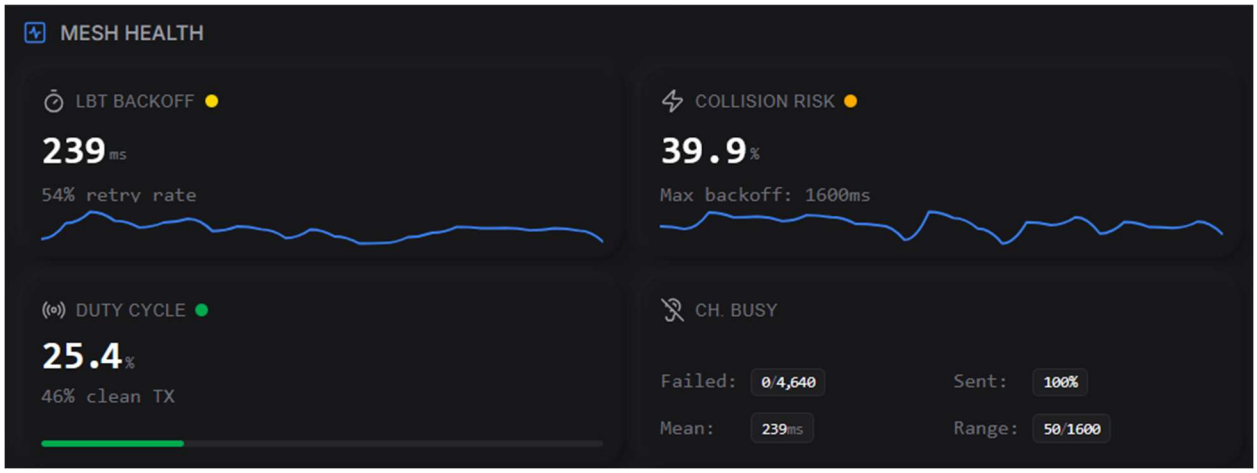
LBT stands for Listen-Before-Talk, a common RF concept where a device must not transmit while the channel is busy. MeshCore does not implement formal LBT, but the radio hardware *naturally* behaves this way because it is half-duplex:

- A node cannot transmit if it is currently receiving a message.
- If a scheduled transmission time arrives while the radio is receiving, the transmission is deferred until the receive operation finishes.

This creates an RX-Busy Deferral window.

RX-Busy Deferral (implicit LBT) This forced delay behaves like an *implicit, topology-dependent random backoff*, since each repeater receives a different set of messages at different times. The result is additional timing decorrelation beyond txdelay and rxdelay, improving collision avoidance and preventing timing lockstep in dense or active meshes. This occurs before *rxdelay* processing occurs. It has been measured on a busy "Hub" repeater.

Figure 3.



RX-busy deferral is only loosely and noisily correlated with neighbor count. It's really a function of:

- who's talking *right now*
- what paths the routing engine chose
- local noise and link asymmetry
- burstiness of traffic

This is an inherent function based on how the radio operates, there is no tuning parameter available. But RX-busy deferral does already implicitly help.

3.8.6 Receive Delay Summary

rxdelay is a lightweight, secondary randomization mechanism that complements *txdelay*. While *txdelay* carries the primary responsibility for collision avoidance, *rxdelay* prevents synchronized receive behavior and timing lockstep, ensuring that the mesh remains statistically decorrelated over time. A modest *rxdelay*—typically 3–4 airtime windows—is sufficient for most environments, with larger values reserved for dense or high-fan-out topologies. LBT Backoff is not being considered.

4. Deployment Guidance

4.0 Density Classification Using SNR-Positive Neighbors

A good indicator of collision risk is the number of neighbors with SNR > 0. This excludes weak or marginal neighbors whose messages are less likely to be decoded and therefore do not meaningfully contribute to contention.

This produces four operational density classes:

Density Class	Neighbor Count (SNR > 0	Interpretation
Sparse	0–3	Few decodable neighbors; low concurrency
Medium	4–8	Typical suburban or rolling-terrain mesh
Dense / Hilltop	9-10	Tight urban clusters, or local elevated sites
Regional	11, 12+	High fan-out, elevated sites

These thresholds are intentionally conservative and map directly to the recommended *txdelay* tiers.

4.1 Sparse Meshes (≤3 neighbors)

- *txdelay* = 1.0–1.2
- *direct.txdelay* = 0.4–0.6
- Produces t = 5–6 for flood, t_direct = 2–3 for direct
- Balanced latency and stability

4.2 Medium Density (4–8 neighbors)

- *txdelay* = 1.2–1.6
- *direct.txdelay* = 0.5–0.8
- Produces t = 6–8 and t_direct = 2–4
- Strong collision reduction

4.3 Dense / Hilltop (9–10 neighbors)

- *txdelay* = 1.8–2.1
- *direct.txdelay* = 0.6–0.9
- Produces t = 9–10+ and t_direct = 3–4
- Maximum desynchronization for flood while keeping direct traffic responsive

4.3 Regional / High-Fan-Out (11–12+ neighbors)

- *txdelay* = 1.8–2.5
- *direct.txdelay* = 0.6–0.9
- Produces t = 9–10+ and t_direct = 3–4
- Maximum desynchronization for flood while keeping direct traffic responsive

5. Operational Symptoms

5.1 Too-Low *txdelay*

- Bursty message loss
- Identical timestamped retransmissions
- Route oscillation

5.2 Too-Low *direct.txdelay*

- Direct messages collide when multiple nodes talk to the same server
- Room traffic shows clumping in logs

5.3 Too-High Values

- Flood: increased end-to-end latency
- Direct: sluggish interactive traffic

6. Stale Neighbor Entries in a Repeaters Neighbor List

There is one additional consideration that need to be addressed: the neighbor count should not include repeaters that are "stale". Repeaters that haven't been heard in some time, because they were moved, their ID changed, they have failed and were not replaced, etc.

- Including them in the neighbor count will unnecessarily skew them to a higher value
- They are no longer active in the neighborhood

6.1 When to Purge Stale Neighbor Entries in a Repeaters Neighbor List

A repeater's neighbor list should remain accurate, minimal, to ensure stable routing and predictable RF behavior. MeshCore uses two independent staleness measurements—Zero-Hop Adverts and Flood Routed Adverts (which are received on the first hop) to mark when they are "heard" and placed/updated in the neighbor table.

Neighbors should receive a Zero-Hop Advert within every 4 hours based on the maximum setting of the Zero-Hop Advert interval. But some may be missed due to low SNR, weather conditions, etc.

- Zero-Hop Advert interval in minutes
    - Valid range: 60–**240**
    - Default 0 (disabled)

It is recommended that any repeater with "Heard" greater than 7 days be purged.

View the neighbor table to determine which are stale and use the CLI command "neighbor.remove (Hex ID)" to delete them. If they are not removed, they will stay until the repeater is rebooted when the neighbor table is cleared and then subsequently rebuilt.

7. Conclusion

txdelay and *direct.txdelay* together form MeshCore's timing randomization framework. *txdelay* governs flood retransmissions and is the dominant factor in collision mitigation, while *direct.txdelay* provides a lighter-weight desynchronization mechanism for direct/unicast traffic.

A *txdelay* or *direct.txdelay* value of 0.6 provides an ~6.25% probability of collision; greater values reduce this further. A collision forces a message to be transmitted twice, increasing airtime consumption. The number of available slots—not airtime factor—is the primary determinant of collision probability:

- **Fewer slots → more collisions**
- **More slots → fewer collisions**

Increasing retransmit delay does not increase airtime consumption; it only delays the transmission. Avoiding collisions, however, **reduces** airtime consumption.

A good factor for selecting values for an individual repeater is the number of neighbors with **SNR > 0**. Raw neighbor count is misleading; low-SNR neighbors often fail to deliver usable messages and should not influence density classification

8.0 Process to Manually Optimize the *rxdelay*, *txdelay,* and *direct.txdelay* Parameters

1. **Remove any Stale Neighbors in the Neighbor Table** (last heard > 7 days)
2. **Count neighbors with SNR > 0.0**
    - Filters out noise-floor ghosts
    - Excludes intermittent or unusable links
    - **NOTE:** A higher SNR threshold may be chosen to eliminate additional repeaters due to a low signal strength. Possibly for higher neighbor counts, use a more stringent criterion.

3. **Use this count as an index into this tuning table**:

Density Class	**Neighbor Count**	***txdelay***	***direct.txdelay***	***rxdelay***
Sparse	0	1.0	0.4	2
	1	1.1	0.5	2
	2	1.2	0.6	3
	3	1.2	0.6	3
Medium	4	1.3	0.7	3
	5	1.4	0.7	3
	6	1.5	0.7	4
	7	1.6	0.8	4
	8	1.7	0.8	4
Dense	9	1.8	0.8	5
	10	1.9	0.9	6
Regional	11	2.0	0.9	7
	12+	**2.1 -2.5**	**0.9**	8

**** Please note the current CLI will not accept a txdelay > 2**

3. **Apply these new values via the CLI commands:**

- set rxdelay {value}
- set txdelay {value}
- set direct.txdelay {value}

Summary:

- The default values for rxdelay, txdelay and direct.txdelay need to be changed.
- Use the procedure to determine the values appropriate for your repeater
- Set these values using the CLI into your repeaters

If everyone adopts these changes, the Mesh performance will improve.

If your neighbor repeaters do not make the change, you will not benefit. But if you adopt the changes, your neighbor will benefit. So, encourage everyone to adopt these recommended changes.