How Should “Message Settings: Direct Message Acks” in a User be Set? (1 or 2)

How ACKs work in MeshCore

A Direct Message (DM) is a routed unicast packet sent from the DM sender to the DM receiver. If the DM’s header has the ACK request bit set, then:

- The DM receiver becomes the ACK sender.
- It emits an ACK flood into its local neighborhood.
- That flood propagates outward until it reaches the DM sender, which becomes the ACK receiver.

MeshCore ACKs are flooded, not routed, so they must be:

- Low airtime to avoid congestion
- Stateless (no path recording)
- Easily suppressible (duplicate detection)
- Fast to rebroadcast (minimal parsing)

The ACK is not routed back along the DM path. It is a probabilistic flood, so its success depends on the RF conditions around both ends.

What does an ACK packet look like:
MeshCore ACK packets are intentionally tiny, fixed-purpose flood packets. They carry only the minimum information needed for the originator to correlate the ACK with an outstanding TX entry.

The exact byte count varies slightly by implementation, but a MeshCore ACK is between 7 and 11 bytes on-air, depending on whether the implementation includes optional hop-limit or link-quality metadata and is dramatically smaller than routed or data packets because ACKs are flooded, **so airtime minimization is critical.**

Required fields in a MeshCore ACK
These fields exist in every ACK, regardless of platform or build options.

1. Packet Type (1 byte)
   Identifies the packet as an ACK.
   Usually a small integer constant (e.g., 0x03).

2. Originator NodeID (2 bytes)
   The NodeID of the node that sent the original message.
   Used by the mesh to suppress rebroadcasts and by the originator to match the ACK.

3. Sequence Number (2 bytes)
    The sequence number of the original message being acknowledged.
    This is the key correlation field for the sender's TX queue.

4. TTL / Hop Limit (1 byte)
    Controls how far the ACK flood propagates.
    Default is usually 3–5 hops depending on mesh density.

5. Flags (1 byte)
    Bitfield indicating:
    • Flood mode
    • Duplicate suppression hints
    • Optional reliability bits

    This is the smallest field but carries important behavior toggles.

Minimum required total: 7 bytes.

Optional fields (implementation-dependent)

Some MeshCore builds include additional metadata to improve passive routing hints or link scoring. These are not required for ACK semantics.

1. RSSI/SNR hint (1–2 bytes)
    A compressed link-quality metric from the node generating the ACK.
    Used only for passive route scoring.

2. Hop Count (1 byte)
    Incremented by each rebroadcasting node.
    Useful for debugging or adaptive TTL logic.

3. Timestamp delta (1 byte)
    Rarely enabled; used for congestion analysis.

    Optional total: 1–4 bytes.

This is the format most MeshCore implementations converge on:

Offset	Size	Field	Description
0	1	type	Packet type = ACK
1	2	originator_id	NodeID of original sender
3	2	seqno	Sequence number of original packet

5	1	ttl	Remaining hop limit
6	1	flags	Flood + suppression flags
7	0–4	optional	RSSI/SNR, hop count, timestamp

Why Properly set rxdelay/txdelay/direct.txdelay Improves Overall DM Success in MeshCore Networks

Directed Messages (DMs) in MeshCore rely on a routed unicast path from the DM sender to the DM receiver, followed by an optional acknowledgement (ACK) flood initiated by the DM receiver. Because DMs and ACKs traverse different RF neighborhoods, their success depends on timing, density, and collision behavior at both ends of the link. Adaptive timing parameters—**rxdelay**, **txdelay**, and **direct.txdelay**—significantly improve DM reliability by reducing synchronized collisions, breaking correlated failures, and improving the probability that both the DM and its ACK complete successfully.

DM and ACK Mechanics

A DM is routed hop-by-hop from the **DM sender** to the **DM receiver**. If the DM requests an ACK, the DM receiver becomes the **ACK sender**, emitting an ACK flood that must reach the **DM sender**, which becomes the **ACK receiver**. The ACK is not routed; it is a probabilistic flood whose success depends on:

- Whether it can escape the ACK sender's neighborhood.
- Whether it can reach the ACK receiver's neighborhood.

Because these neighborhoods differ, timing parameters must mitigate collision and synchronization effects at both ends.

Environmental Regimes and Their Impact

MeshCore networks operate across several broad density regimes:

- **Sparse (0–2 neighbors):** Failures occur near the ACK receiver due to weak final-hop reachability.
- **Medium (5–8 neighbors):** Failures are semi-random; both ends contribute to loss.
- **Dense (9+ neighbors):** Failures occur near the ACK sender due to synchronized collisions.

rxdelay

rxdelay can set receive-window timing based on local neighbor density. This improves DM and ACK success by:

- Desynchronizing receive windows across nodes.
- Reducing simultaneous rebroadcast attempts.
- Lowering the probability of ACK-storm overlap.
- Improving DM forwarding reliability before the ACK stage.

By spreading out receive windows, *rxdelay* reduces the likelihood that multiple nodes attempt to process or forward the same packet at the same moment, lowering collision probability.

txdelay

Why not setting *txdelay* manually will cause essentially a guaranteed collsion for both the first and second ACK attempts. The default value of txdelay is 0.5 which only generates three random backoff slots. In a Dense environment that has a large number of neighbors is very probable that a collision will occur whether 1 or two ACKs are set. This is why the *txdelay* should be changed to reduce collisions by raising the value of *txdelay* to reflect the number of neighbors.

txdelay introduces jitter into transmit timing based on neighbor count. This improves reliability by:

- Reducing synchronized transmissions in dense neighborhoods.
- Breaking correlation between repeated failures.
- Allowing ACK floods to escape the ACK sender's environment more reliably.
- Improving DM forwarding by reducing deterministic collision patterns.

In medium-density environments, *txdelay* increases the independence of loss events, improving the probability that at least one transmission succeeds.

direct.txdelay

direct.txdelay governs spacing between direct-mode DM retries. It improves DM delivery by:

- Preventing deterministic timing loops where retries collide with each other.
- Allowing retries to land in different RF conditions.
- Improving sparse-mesh performance where reachability varies over short time intervals.

direct.txdelay enhances DM success before ACKs are even involved, making it a critical component of end-to-end reliability.

ACK Count Behavior

The ACK sender chooses whether to transmit 1 or 2 ACKs. However, the ACK's success depends heavily on both ACK Sender and ACK receiver's environment. The behavior across density regimes is:

- **Sparse:** Both ACKs fail for the same structural reason; 2 ACKs do not help.
- **Medium:** Losses are semi-random; 2 ACKs can help.
- **Dense:** Failures are correlated; 2 ACKs do not help.

Because the DM sender cannot know the DM receiver's environment, and because sparse and dense regimes both punish 2 ACKs, **1 ACK is the best setting reflecting all environment types**.

Impact of Updated Timing

rxdelay, *txdelay*, and *direct.txdelay* collectively improve DM success by:

- Reducing synchronized collisions.
- Increasing temporal diversity across transmissions.
- Improving DM forwarding reliability.
- Improving ACK escape probability.
- Reducing correlated failures in dense environments.
- Improving retry effectiveness in sparse environments.

These mechanisms enhance DM reliability across all density regimes, even though ACK count cannot be safely adapted without a dynamic mechanism.

How each environment affects ACK success

Environment of the DM receiver (also the ACK sender): This environment determines whether the ACK flood can escape the receiver's neighborhood.

• Many neighbors rebroadcasting → collisions near the ACK sender.
• High airtime load → ACK #1 and ACK #2 fail for the same reason.
• Dense meshes → correlated failures (both ACKs die together).

In dense or medium dense meshes, most ACK failures happen near the ACK sender.

Environment of the DM sender (also the ACK receiver): This environment determines whether the ACK flood can reach the DM sender on the final hop.

- Few neighbors → weak fan in for the ACK flood.
- Shadowing or edge of mesh geometry → ACK dies on the last hop.
- Sparse meshes → structural reachability limits dominate.

In sparse meshes, most ACK failures happen near the ACK receiver.

How 1 vs 2 ACKs behave across mesh densities

DM sender env	DM receiver env	Best	Why
Sparse	Sparse	1	Failures are structural; 2 ACKs don't help
Sparse	Medium	1	Final hop is weak; 2 ACKs don't help
Sparse	Dense	1	Dense side causes correlated failures
Medium	Sparse	1	Sparse side dominates; 2 ACKs don't help
Medium	Medium	2	Losses are independent; 2 ACKs help
Medium	Dense	1	Dense side causes correlated failures
Dense	Sparse	1	Sparse side dominates
Dense	Medium	1	Dense side dominates
Dense	Dense	1	Both ACKs fail together

Only medium–medium produces independent loss events.

Because the DM sender cannot know whether the DM receiver is in a sparse, medium, or dense region:

- Sparse meshes punish 2 ACKs.
- Dense meshes punish 2 ACKs.
- Medium meshes benefit from 2 ACKs.

Two out of three regimes make 2 ACKs worse.

Bottom line
If the DM sender somehow knows that both ends are medium density, then:

2 ACKs is the better choice. But they would need to change this with every DM sent if the recipients are in dissimilar densities (it's not practical to do).

Plus, because MeshCore does not expose the DM receiver's environment to the DM sender, this situation cannot be detected automatically today — which is why **1 ACK is the correct default value and why it is recommended to not be changed.**

Conclusion

Adaptive *rxdelay*, *txdelay*, and *direct.txdelay* significantly improve end-to-end DM success in MeshCore networks by reducing collision probability, breaking deterministic timing patterns, and improving both DM forwarding and ACK return reliability. While medium-density environments benefit from 2 ACKs, the DM sender cannot reliably detect the DM receiver's environment. Without a dynamic, protocol-level mechanism to negotiate ACK count, **1 ACK is the only correct setting**, and updated timing parameters are the primary tool for improving DM reliability across all mesh conditions.