Technical FAQ — Halo

About Halo

What is Halo?

Halo is Lagrange's coordination engine for autonomous systems. It's the layer of the autonomy stack that keeps distributed autonomous systems — drones, robots, uncrewed vessels — agreeing on what's true, recovering from failures, and operating without a central controller, even when conditions are bad and getting worse.

Why a coordination layer, why now?

Because the constraint on autonomous systems is no longer the platforms. Ukraine's defense industry can now build more than 8 million FPV drones a year; Russia is planning for more than 7 million in 2026. The bottleneck is the connective tissue: keeping heterogeneous, partially-connected, partially-degraded autonomous systems coordinated under real conditions. Today, every vendor rebuilds some version of that logic inside their own vertical stack. Halo is the substrate so they can stop.

How is Halo different from a mission-autonomy platform like Lattice or Hivemind?

Those platforms bundle coordination with mission behavior — they decide what to fly, what to sense, what to do. Halo doesn't. It owns the layer underneath: shared state, agreement, failure recovery, role coordination. The mission stays the customer's. In practice you can run Halo underneath an existing mission stack — including stacks built on those platforms — without rewriting your application logic.

The mental model: Stripe shipped payments primitives so every team didn't have to rebuild a card processor. Kafka shipped data-stream primitives so every team didn't have to rebuild a log. Halo ships coordination primitives so every autonomy team doesn't have to rebuild consensus, failure detection, and reconfiguration inside their own stack.

How is Halo different from an autopilot or flight-controller stack?

An autopilot controls one platform. Halo coordinates many. They live at different layers and they're complementary, not competitive.

Architecture

Where does Halo sit in the stack?

Three layers, bottom up:

1. Hardware — drones, sensors, communications. Yours and your partners'.
2. **Halo** — the coordination engine. Agreement across distributed agents, decentralized operations, dynamic reconfiguration, redundancy management, and optional reference coordination modes.
3. Application logic — mission execution, customer-defined behavior.

What ships at launch?

Four things:

- **The core coordination engine** — shared-state agreement, failure detection, dynamic reconfiguration, with no central controller.
- **Four reference coordination modes** — collaborative sensing and ISR, communications relay, resilient

formation and area coverage, and task reassignment with asset-loss recovery.
- **Developer interfaces (SDK + APIs)** — drops into the autonomy stacks teams already run. MOSA-compatible. No runtime replacement required.
- **A live, browser-based sandbox** — drop assets onto a map, inject failure conditions, and watch the system hold shape in real time. Available at `playground.lagrange.dev`.

Halo runs at the edge on the platform, off-platform on a ground station or command node, or in a hybrid configuration — consistent with how real operators deploy.

What's actually in Halo?

- Shared-state agreement and synchronization
- Node-failure detection (phi-accrual-style, tuned for partial connectivity)
- Dynamic reconfiguration and role reassignment
- Group membership and leader-election primitives
- Partition handling and recovery
- Network-state and node-health observability

How does Halo decide what to coordinate on, given the network is unreliable?

The architecture is layered the way real operations are layered.

Swarm-wide decisions — repositioning, role assignment, anything that needs the whole group to agree — run through full consensus across the network. Slower, but durable.

Active-neighborhood coordination — inside each node's reachable subset of the swarm, coordination continues when the larger network is partitioned. Split-resilient by design. Two halves of a fragmented swarm can each keep operating, and reconcile when they reconverge.

Local cached state — every node holds the latest known state it has. So when a drone has to evade a threat or avoid a collision in 200 milliseconds, the decision happens on the drone, with the data it already has, without waiting on a vote.

The three tiers are how Halo stays responsive when consensus is slow, partition-tolerant when the swarm fragments, and durable when the network heals.

What are "reference coordination modes" and are they apps?

They're coordination *patterns*, not apps. Four ship with Halo as opt-in templates:

- Collaborative sensing & ISR
- Communications relay
- Resilient formation & area coverage
- Task reassignment & asset-loss recovery

The core engine is fully usable without them. We ship the modes because operators keep asking for the same patterns and we want integration to be fast — not because we're trying to write the mission for you.

Does Halo work without a central controller?

Yes. There is no required central controller. Halo maintains agreement and keeps operating when central control is unavailable, partitioned, or destroyed. Off-platform components — ground stations, supervision dashboards, operator consoles — are supported but optional. None are on the critical path.

Operating conditions

Does Halo require GPS?

No. Halo doesn't depend on GNSS for its own operation. The architecture is designed for the GPS-denied environment that's now the default along the 1,200-kilometer Ukrainian front. Customer applications that rely on GPS still rely on GPS — that's not Halo's problem to solve.

How does Halo handle time synchronization without GPS?

Halo's shared-state protocol does not require synchronized wall-clock time across nodes. State updates carry vector-clock-style causal context, so receivers can determine update ordering — and detect concurrent updates — without relying on agreement about absolute time. This is the design choice that lets the system operate when GNSS is unavailable, jammed, or spoofed.

What network conditions does Halo tolerate?

Designed for partial, time-varying connectivity with bursty latency, packet loss, and intermittent partitions. The shared-state protocol is opportunistic and causally consistent — nodes converge as the network heals, rather than blocking on synchronous agreement. Specific tolerance numbers (latency budget, loss rate, partition duration) are workload-dependent; we publish reference benchmarks with the sandbox.

How does Halo behave under jamming?

Jamming surfaces two ways: silence (fewer messages) and spoofing (false messages).

Halo handles silence with a graded suspicion-score failure detector that avoids the role-thrashing failure mode of timeout-based detectors. A drone that goes quiet doesn't get declared dead the second the radio gets noisy.

It handles spoofing with authenticated, replay-resistant liveness signals, and by cross-checking presence claims against actual causal activity in the shared-state layer. A node that "looks alive" by ping but isn't doing anything in the shared state is itself a suspicion signal.

How does Halo keep a drone responsive during threat evasion?

It doesn't make the drone call home. Every node holds the most recent agreed state locally, so decisions that have to happen in tens or hundreds of milliseconds — evading a threat, avoiding a collision, breaking formation — happen on the drone with the data it already has. Halo's job is to ensure that local state is as fresh and consistent as the network allows, not to gate every decision behind global agreement.

How does Halo scale?

The launch sandbox demonstrates 10–20 nodes. The architecture is designed for higher scale, and the active-neighborhood model is the primitive that gets it there — Halo doesn't require a flat global mesh of agreement. Specific scaling envelopes ship with subsequent benchmarks; the +3-month milestone takes us to 50 nodes with public scorecards per release.

Integration & deployment

What's the deployment model — edge or cloud?

Hybrid. Halo runs at the edge (on the platform) and off-platform (on a ground station or command node). The split is mission- and platform-dependent. Edge deployment is what enables continued operation when comms with the ground are gone.

How does Halo integrate with my existing stack?

Through stable, narrow APIs and SDKs. The surface is intentionally small — publish a state key, subscribe to a key, watch for changes, request a role assignment, observe membership and health. Designed for integration without requiring a new runtime environment or a rewrite of customer application logic.

We're realistic about this. No middleware integration is truly "plug-and-play." We'd rather publish honest reference integrations and integration guides than overpromise on top.

What language and runtime is the SDK? What OSes does it run on?

The SDK is built for the runtimes autonomy teams actually ship — Linux on the platforms most drone OEMs run today, with bindings for the languages those teams use to write mission logic. The full SDK reference,

with supported languages, OS targets, and architecture-specific notes (x86_64, ARM64, common embedded targets), publishes ahead of the +3-month field trial. If you have a specific platform constraint we should know about, talk to us — we'd rather size the fit honestly than guess publicly.

Is Halo MOSA-compatible?

Yes. Halo's developer interfaces are designed to fit the Modular Open Systems Approach: stable, narrow APIs at the SDK level, no runtime replacement, no required orchestration plane. The intent is to make Halo a coordination substrate that integrates cleanly with open-architecture programs rather than a closed stack that has to be adopted whole.

What about bandwidth and observability?

Bandwidth: typed and partitioned. Not all swarm state has the same consistency requirements — identity and group membership need strong agreement; sensor fusion is allowed to converge over time; operational telemetry can be eventually consistent. Treating all state with the same protocol is the cheapest way to waste a bandwidth budget, so Halo exposes typed primitives that let the application layer be explicit about what it needs. Specific per-node bandwidth envelopes are workload-dependent and ship with the reference benchmarks.

Observability: Halo exposes network-state and node-health as first-class signals — what the swarm believes, who is reachable, who is degrading, and why. The +3-month operator-console release surfaces this as a read-only view into coordination state. Observability, not command-and-control.

What about heterogeneous platforms — different OEMs, different OSes?

Platform-agnostic by design. Halo isn't tied to a specific autopilot, OS, or vendor. Integration work scales with the diversity of the fleet. The substrate doesn't change.

Cryptography & roadmap

Where does cryptography come in?

Today, Halo uses authenticated channels for liveness and state updates. The roadmap extends Halo with cryptographic verification — the ability to produce proofs of what the swarm decided, observed, and did. That work bridges directly into Lagrange's broader verifiable-autonomy thesis. Specifics on the verifiable-coordination roadmap are published separately.

What's on the roadmap?

The published roadmap runs over five horizons. The full version, with calendar markers and program inflection points, lives at `lagrange.dev/roadmap`. The headline pieces:

- **H · 00 · Today (Q2 2026) — LAUNCH.** Halo ships. Four reference modes. Layered coordination architecture. Edge + ground deployment, SDK-level integration, MOSA-compatible.
- **H · 01 · +3 months — NEXT.** First field trial. Scale up to 50 nodes. Operator console. Coordination SLA published. SDK adapters and reference-platform program with select drone OEMs.
- **H · 02 · +6 months — BREAKTHROUGH.** Adaptive strategic coordination. Resilient swarm-to-base relay. Anti-drone evasion. Multi-vendor swarms. Multi-domain consistency across air, ground, and maritime surface. Field validation under EW and GPS denial.
- **H · 03 · +12 months — BREAKTHROUGH.** AI-driven coordination. GPS-less coordination (peer-to-peer localization, mesh navigation). Stealth mode (coordinated low-RF-emission posture). Counter-swarm coordination. Cross-domain coordination across air, ground, maritime surface, undersea, space.
- **H · 04 · +18 months — EXPLORATION.** Coordinated autonomy at scale (thousand-node formations on a peer-to-peer substrate). Distributed intelligence at the swarm level. Counter-swarm at theater scale. Theater-wide signature management. Adaptive coalition operations. Beyond defense.

Demo, team & getting started

Is the sandbox a real demo?

Yes. The launch sandbox is a working simulation of 10–20 nodes operating in a coordinated mission, surviving comms loss, jamming, and node loss. It's a real running system with a visible operational interface — not a marketing animation. Real-hardware demonstrations follow as their own moment, when the engineering is ready.

Who is Halo for?

Defense primes, drone OEMs, autonomy and robotics teams, government and allied program offices, multi-domain integrators. Anyone building, integrating, or buying autonomous systems that have to coordinate under conditions worse than the lab.

Who built Halo?

Lagrange. Led by Ismael Hishon-Rezaizadeh (CEO), with Charalampos Papamanthou and a broader team of cryptographers and distributed-systems engineers. Lagrange is a member of the Lockheed Martin supplier ecosystem.

How do I get started?

Read the technical overview at **lagrange.dev/defense**. Run the sandbox at **playground.lagrange.dev**. See the full roadmap at **lagrange.dev/roadmap**. Reach out: **defense@lagrange.dev**.