

RFC: Edge Gateway Mode (VPP Site Controller)

Status

Draft

Motivation

The current MVP runs one **edge_agent** per DER asset. That is excellent for testing and early integration, but real deployments typically want a **single on-site gateway** that:

- Maintains one outbound connection to the headend (simpler networking + cert management)
- Represents multiple DERs at that site (PV, BESS, meter, genset, etc.)
- Provides a natural boundary for local safety/coordination and future “VPP-like” aggregation

This RFC defines **Edge Gateway Mode**: one agent process multiplexes telemetry and control for many assets over a single gRPC stream.

Why gRPC for edge connectivity

Edge Gateway Mode relies on a **long-lived, outbound-initiated, bidirectional** connection between site and headend. gRPC (HTTP/2 + protobuf) is a strong fit for this pattern:

Strengths

- **Bidirectional streaming**: telemetry flows agent→headend while setpoints flow headend→agent over the same established stream (no polling/webhooks).
- **Typed contracts**: protobuf schemas reduce integration ambiguity and make message evolution safer (additive fields, defaults).
- **Efficiency**: binary encoding and HTTP/2 multiplexing reduce overhead vs JSON/REST at higher message rates.
- **Flow control**: HTTP/2 provides built-in flow control, enabling bounded buffering and clearer backpressure behavior.

Weaknesses / operational caveats

- **Harder to debug than REST**: operators need **grpcurl** (or equivalent) plus good logs/metrics.
- **Proxy/LB timeouts**: some L7 proxies and load balancers enforce idle timeouts; keepalives and timeout tuning are required.
- **Retry semantics are subtle**: reconnects can duplicate in-flight messages unless messages carry ids/sequence numbers and are handled idempotently.
- **Browser clients**: direct browser access requires gRPC-Web or an HTTP gateway (not a concern for site agents).

Practical guidance (V1 expectations)

- Define and document **keepalive / idle timeout** settings for agent and headend.

- Add **message ids / sequence numbers** (telemetry + setpoints) early to support safe retries and auditability.
- Maintain a small operator toolset: `grpcurl` examples, connectivity checks, and clear log messages for connect/disconnect reasons.

Goals

1. **One outbound connection per site** (gateway agent) supporting N logical DER assets.
2. Preserve **per-asset identity** at the headend (`asset_id` remains the control/telemetry address).
3. Support both:
 - **Per-asset** dispatch (`asset_id` targeted)
 - Optional **group dispatch** (e.g., `vpp_id` / `site_id`) with simple split policy
4. Keep behavior well-defined for reconnects and offline periods.
5. Minimize protocol churn and maintain backwards compatibility where practical.

Non-goals (for V1)

- Optimization / constraint-aware allocation (OPF, feeder limits, Volt/VAR)
- Program orchestration (DR events, schedules, market programs)
- Full DER adapter framework (vendor-specific protocols), beyond a clean interface boundary
- Multi-region routing / broker mesh
- Hard real-time control guarantees

Terminology

- **Gateway agent**: one `edge_agent` instance running on-site, managing multiple assets.
- **Logical asset**: a single controllable/measured DER represented by an `asset_id`.
- **Site**: a physical location containing multiple logical assets.
- **Group**: a collection of assets for “aggregate dispatch” (site, VPP, portfolio segment).
- **Outbound-initiated connection**: the agent **initiates** the network connection to the headend (no inbound listener/port required on the site). The headend can still send commands back **over the same established stream**.

Proposed architecture

Current (MVP)

- One gRPC stream per `edge_agent`
- One `edge_agent` == one `asset_id`
- The gRPC stream is **outbound-initiated** by the agent (the site does not expose an inbound port for control).
- Commands/setpoints flow headend→agent **over the same stream** the agent created.

Edge Gateway Mode (V1)

- One gRPC stream per gateway agent
- One gateway agent == **many** `asset_ids`
- Headend routing: `asset_id` → `gateway_session` → `gRPC stream`

Data model additions

- `site_id` (stable identifier for a site/gateway; may already exist implicitly in `assets.yaml`)
- `asset_id` remains the stable ID for each logical DER
- Optional: `group_id` / `vpp_id` for aggregate dispatch targeting

Minimal viable representation:

- Each asset has `{ asset_id, site_id, capacity_kw/mw (optional), type (optional) }`

Protocol changes (gRPC)

High-level rule

Every message that is per-asset must carry `asset_id`.

Telemetry

- MUST include `asset_id`
- MAY include per-asset metadata (quality/status) as fields evolve

Setpoint / Dispatch Command

- MUST include either:
 - `asset_id` (per-asset command), OR
 - `group_id` / `site_id` (aggregate command)
- If group-targeted, the gateway agent is responsible for splitting into per-asset actions.

Registration

Two compatible patterns are acceptable; pick one and standardize: A) **Gateway registers once** with a list of assets:

- `{ site_id, assets: [asset_id, ...], capabilities(optional) }`

B) **Gateway registers each asset** on the same stream:

- multiple Register messages, one per `asset_id`, each tagged with `site_id`

Recommendation: **Pattern A** (single register with asset list) for lower overhead and clear ownership.

Backward compatibility

- If the headend receives Telemetry without `asset_id`, treat it as legacy single-asset mode (use the session's associated default `asset_id`).
- If the agent receives Setpoint without `asset_id`, treat it as legacy single-asset mode.

Dispatch splitting policy (gateway responsibilities)

If the headend targets a group (`site_id / vpp_id`), the gateway agent applies a **deterministic split policy**.

V1 recommended default: **proportional-to-capacity**

- Each asset i gets: $mw_i = mw_total * (cap_i / sum(cap))$
- Apply clamps per asset (min/max) if available
- Deterministic rounding rule (e.g., round to 0.001 MW, distribute remainder by largest fractional part)

Fallback if no capacities:

- Equal split across online assets

Additional V1 policy (optional):

- Priority order: BESS first, then PV curtailment, etc. (only if customer needs it)

State and lifecycle

Gateway session state

Headend maintains:

- gateway session: `{ gateway_id/site_id, connected, last_seen, assets_served[] }`
- per-asset presence derived from gateway session + per-asset heartbeats (optional)

Offline behavior

- Per-asset dispatch created while gateway is offline:
 - Stored as pending by `asset_id` (or by group, expanded into per-asset pending)
 - Delivered on reconnect subject to TTL/expiry (recommended in V1)
- Telemetry while offline:
 - V1: best-effort (no buffering guarantee), unless explicitly implemented
 - Later: store-and-forward spool on gateway

Security implications (V1 expectation)

- One gateway cert/identity per site is acceptable for V1.
- Network posture: the site typically only needs **outbound TCP** access to the headend on the chosen port; **no inbound firewall/NAT rule** is required for headend→agent commands because commands flow back over the established stream.
- Headend authorization should enforce:
 - A gateway may only claim assets it is allowed to serve (asset allow-list).
- Later: per-asset credentials, delegated identities, and tighter claims.

Observability

Gateway agent should emit:

- Per-asset telemetry rate
- Per-asset last successful setpoint apply time

- Group dispatch split results (log/trace), including final per-asset allocations

Headend should expose:

- Connected gateways and assets served
- Dispatch success/ack rates (once ack exists)
- Telemetry lag (now - last telemetry timestamp) per asset

Implementation plan (suggested sequence)

The goal is to deliver Edge Gateway Mode in small, testable increments without breaking the existing single-asset agent.

1) Proto updates (add addressing fields)

What you change

- Update `agent.proto` so that every per-asset message can be routed unambiguously.

Minimum fields

- Add `asset_id` to Telemetry messages.
- Add `asset_id` to Setpoint/Dispatch messages.
- Add `site_id` plus an `assets[]` list to the Registration message (Pattern A).

Rules to implement

- Backward compatibility: if `asset_id` is missing, treat as legacy single-asset mode.

How to test

- Compile protobufs for both headend and agent.
- Run the existing single-asset agent unchanged (or with `asset_id` defaulting) to confirm no regressions.

2) Headend: session registry and routing by `asset_id`

What you change

- Extend the headend's in-memory connection/session registry to map:
 - `site_id` -> `gateway_session`
 - `asset_id` -> `gateway_session` (or `asset_id` -> `stream handle`)

Behavioral requirements

- On registration, store the `site_id` and the list of served `asset_ids`.
- When a Setpoint/Dispatch is created for a specific `asset_id`, the headend must find the correct connected gateway stream and write the Setpoint onto it.

How to test

- Unit test the routing map logic.
- Manual test: single-asset agent registers and still receives setpoints.

3) Agent: internal model becomes a collection (multi-asset runtime)

What you change

- Refactor the agent from “one asset state” to a `HashMap<asset_id, AssetRuntime>`.
- Each `AssetRuntime` holds the state needed to generate telemetry and apply setpoints (current mw, limits, last telemetry ts, etc.).

Design guidance

- Keep the legacy single-asset path working by initializing the map with one entry.
- Treat “gateway mode” as a configuration option (e.g., one site with N assets).

How to test

- Unit test `AssetRuntime` tick/update logic.
- Add a small in-process test that runs N assets for a few ticks and produces N telemetry messages.

4) Telemetry mux: send per-asset telemetry over one stream

What you change

- Modify the agent send loop to iterate all `AssetRuntime` entries and emit telemetry tagged with `asset_id`.
- Keep the existing cadence (e.g., every ~4s), but define whether you:
 - send all assets each tick, or
 - round-robin assets across ticks (later optimization)

Behavioral requirements

- Each telemetry message MUST include `asset_id`.
- Telemetry timestamps should reflect per-asset generation time.

How to test

- Headend `GET /telemetry/{asset_id}` returns distinct values per asset.
- Verify telemetry does not “collapse” into a single asset due to routing bugs.

5) Dispatch routing in the agent

What you change

- Update the agent receive loop to handle two cases:
 1. **Per-asset setpoint:** if `asset_id` is present, apply only to that `AssetRuntime`.
 2. **Group/site setpoint:** if `site_id/group_id` is present, compute a split and then apply per asset.

Minimum viable split policy (V1 default)

- Proportional-to-capacity when capacities are provided; otherwise equal split.
- Deterministic rounding and clear logging of the final allocations.

How to test

- Issue a dispatch for one `asset_id` and confirm only that asset changes telemetry.
- Issue a group dispatch and confirm the computed split is stable and sums to the requested total.

6) Docs + test harness updates

What you change

- Update the example `assets.yaml` to show a multi-asset site (one `site_id`, multiple `asset_ids`).
- Extend `agent_launcher` (test-only) to optionally start:
 - legacy mode (one agent per asset), and
 - gateway mode (one agent serving multiple assets)

How to test

- Local bring-up: run headend + one gateway agent serving multiple assets.
- Confirm REST endpoints work per asset and dispatch works for both per-asset and group targeting.

Suggested "Codex-friendly" incremental prompts

When you ask Codex to implement this, keep requests narrowly scoped. The steps below are grouped by milestone.

A) Minimal Edge Gateway Mode (demo-ready)

1. **Proto addressing:** "Update `agent.proto` to add `asset_id` fields to Telemetry and Setpoint; add `site_id + assets []` to Register; regenerate code; keep backward compatibility."
2. **Headend routing map:** "In headend, add a map from `asset_id` to the active gRPC stream based on the registration `assets []` list; ensure legacy single-asset still works."
3. **Agent multi-asset state:** "Refactor `edge_agent` state into `HashMap<asset_id, AssetRuntime>` but keep legacy single-asset behavior working."
4. **Telemetry multiplexing:** "Modify the agent telemetry loop to emit one telemetry message per `asset_id` each tick; ensure headend stores latest telemetry per asset."
5. **Per-asset dispatch:** "Update Setpoint handling so messages with `asset_id` only apply to that asset; confirm `/dispatch` works per asset."
6. **Optional group dispatch split:** "Add support for group/site setpoints (e.g., `site_id`) and implement a deterministic split policy (proportional-to-capacity, else equal split) with clear logging."

B) V1 Pilot Hardening (production trial)

7. **REST + validation semantics:** "Extend `/dispatch` to accept either `asset_id` or `site_id/group_id`; add request validation and clear error responses; document behavior."
8. **Persistence/schema updates (if DB enabled):** "Update DB schema and queries to persist `site_id` and per-asset telemetry/dispatch keyed by `asset_id`; add indexes for `(asset_id, ts)`."
9. **gRPC keepalive/timeout tuning:** "Add explicit keepalive/idle timeout configuration to agent and headend; document recommended load balancer/proxy settings and defaults."
10. **Message ids / sequencing:** "Add `setpoint_id` (command id) and `telemetry_seq` to messages; ensure headend logs and stores them for audit; handle duplicate delivery safely."
11. **Dispatch acknowledgements & lifecycle:** "Add agent→headend SetpointAck messages (received/applied/failed + timestamp); update headend to expose dispatch state and success rate."

12. **Authorize asset claims:** "On registration, verify the gateway is allowed to claim each `asset_id` (config/DB allow-list); reject/flag unauthorized claims."
13. **Observability & runbooks:** "Add metrics for connected gateways, per-asset telemetry rate/lag, dispatch success; add structured logs for connect/disconnect reasons; write a short pilot runbook."

C) Optional V1+ Enhancements (only if needed)

14. **Offline queue with TTL:** "Implement a durable pending-dispatch queue with TTL/expiry semantics and deterministic delivery order on reconnect."
15. **Store-and-forward telemetry (gateway spool):** "Add optional on-site buffering for telemetry during outages, with bounded disk use and replay rules."
16. **Per-asset identities:** "Support per-asset credentials or delegated identities to tighten authorization beyond one cert per gateway."

Acceptance criteria

- One gateway agent can serve at least 10 assets with a single gRPC stream.
- `GET /telemetry/{asset_id}` returns the correct asset's latest telemetry.
- `POST /dispatch` targeting an `asset_id` affects only that asset.
- Optional: group dispatch splits deterministically and logs the split outcome.

Open questions

- Do we represent groups as `site_id` only in V1, or introduce `vpp_id` immediately?
- Do we require per-asset heartbeats, or is gateway heartbeat sufficient for V1?
- Do we enforce "asset claims" at registration time (recommended), and where is the source of truth (DB vs config)?