integrated_system/docs/05_TEST_PLAN_GATEWAY_MODE.md

Test Plan: Edge Gateway Mode (demo-ready)

This test plan verifies **A) Minimal Edge Gateway Mode (demo-ready)**:

- One **gateway `edge_agent`** maintains **one outbound-initiated bidirectional gRPC stream** to `der_headend`
- That single stream serves **multiple `asset_id`s**
- Headend tracks **distinct latest telemetry per asset**
- **Per-asset dispatch** affects only the targeted asset
- Optional: **site/group dispatch** is split deterministically (only if implemented)

Prerequisites

How many `edge_agent` processes should be running?

This test plan covers **Edge Gateway Mode** (one agent serving multiple assets) as the target design, but the current codebase may still operate in **single-asset mode**.

- **Current (single-asset mode today):** run **one `edge_agent` per asset** in `assets_test.yaml`.
- **Gateway Mode (after implementation):** run **one `edge_agent` per site** (one agent serves multiple `asset_id`s behind a `site_id`).

If you are using `agent_launcher` (test harness), it will spawn **one agent per asset**.

Services

- `der_headend` running with REST on `http://127.0.0.1:3001`
- `der_headend` running with gRPC on `127.0.0.1:<grpc-port>` (default is often `50051`; your launcher may default to `50070`)
- One or more `edge_agent` processes connected to the headend (either one-per-asset **or** one-per-site in gateway mode)

Start commands (local)

Assumption: `assets_test.yaml` is located at `der_headend/assets_test.yaml` from the repo root.

Terminal 1 — start headend

```
cd /Users/pfriedland/projects/rust/integrated_system
export ASSETS_PATH=der_headend/assets_test.yaml

# Pick a gRPC port and use it consistently. If you plan to use
```

```
agent_launcher,
# setting HEADEND_GRPC_ADDR to 50070 avoids the common 50051 vs 50070
mismatch.
export HEADEND_GRPC_ADDR=127.0.0.1:50070

cargo run -p der_headend
```

Terminal 2 — start agents (recommended: launcher test harness)

```
cd /Users/pfriedland/projects/rust/integrated_system
export ASSETS_PATH=der_headend/assets_test.yaml
export HEADEND_GRPC=127.0.0.1:50070

cargo run -p agent_launcher
```

Sanity check

```
curl -s http://127.0.0.1:3001/agents | jq .
```

If `/agents` is empty, the most common cause is a **gRPC port mismatch** between `HEADEND_GRPC_ADDR` (headend) and `HEADEND_GRPC` (agents).

-### API basics

- REST dispatch example (per-asset):

  ```
  curl -X POST http://127.0.0.1:3001/dispatch \
    -H 'content-type: application/json' \
    -d '{"asset_id":"7ba7b810-9dad-11d1-80b4-00c04fd430c8","mw":5}'
  ```

Helpful tooling (optional)

- `jq` for pretty printing JSON:

  ```
  brew install jq
  ```

Reference asset IDs for this run

Pick at least **two** assets that the gateway agent claims it serves.

Example:

- ASSET_A = `7ba7b810-9dad-11d1-80b4-00c04fd430c8`

- ASSET_B = `<another-asset-uuid-from-assets.yaml>`

For convenience:

```
export HEADEND=http://127.0.0.1:3001
export SITE_ID=6ba7b810-9dad-11d1-80b4-00c04fd430c8
export ASSET_A=7ba7b810-9dad-11d1-80b4-00c04fd430c8
export ASSET_B=8ba7b810-9dad-11d1-80b4-00c04fd430c8
```

A1–A2: Registration + headend routing map

Goal

Headend receives a Register message indicating the gateway serves `[ASSET_A, ASSET_B, ...]`, and can route setpoints by `asset_id` to that stream.

Actions

1. Start `der_headend`
2. Start gateway-mode `edge_agent`
3. Check connected agent list (best-effort presence):

   ```
   curl -s $HEADEND/agents | jq .
   ```

4. Confirm assets are known to headend:

   ```
   curl -s $HEADEND/assets | jq .
   ```

Expected results

- `/agents` shows **one** connected session (or equivalent)
- `/assets` includes `ASSET_A` and `ASSET_B`
- Headend logs (recommended) show something like:
  - `Register site_id=... gateway_id=... assets=[ASSET_A, ASSET_B, ...]`

Common failures

- `/agents` empty: agent not connected / gRPC port mismatch / TLS mismatch
- `/assets` missing IDs: assets.yaml not loaded or headend uses a different config file

---

A3–A4: Telemetry multiplexing (distinct telemetry per asset)

Goal

Each `asset_id` has its **own** latest telemetry record.

Actions

1. Fetch latest telemetry for each asset:

   ```
   curl -s $HEADEND/telemetry/$ASSET_A | jq .
   curl -s $HEADEND/telemetry/$ASSET_B | jq .
   ```

2. Wait ~5–10 seconds, then fetch again:

   ```
   sleep 6
   curl -s $HEADEND/telemetry/$ASSET_A | jq .
   curl -s $HEADEND/telemetry/$ASSET_B | jq .
   ```

Expected results

- Both endpoints return JSON successfully (not 404)
- Timestamps (or last-updated indicators) advance over time
- Values are **not accidentally identical across all assets** unless your simulator intentionally makes them identical

Red flags (classic gateway bugs)

- Both assets show identical values and identical timestamps forever
  - Likely overwriting a single “latest telemetry” slot instead of keying by `asset_id`
- One asset updates, the other returns 404
  - Likely telemetry missing `asset_id` or headend routing map incomplete

---

A5: Per-asset dispatch (must only affect the targeted asset)

Goal

Dispatch to `ASSET_A` changes only `ASSET_A` behavior/telemetry.

Actions

1. Record baseline telemetry:

   ```
   echo "Baseline A:"
   curl -s $HEADEND/telemetry/$ASSET_A | jq .

   echo "Baseline B:"
   curl -s $HEADEND/telemetry/$ASSET_B | jq .
   ```

2. Dispatch to **ASSET_A**:

```
curl -s -X POST $HEADEND/dispatch \
  -H 'content-type: application/json' \
  -d "{\"asset_id\":\"$ASSET_A\",\"mw\":5}" | jq .
```

3. Wait 1–2 ticks (10 seconds is safe), then re-check telemetry:

```
sleep 10
echo "After dispatch A:"
curl -s $HEADEND/telemetry/$ASSET_A | jq .

echo "After dispatch B:"
curl -s $HEADEND/telemetry/$ASSET_B | jq .
```

Expected results

- Telemetry for `ASSET_A` shows a clear change consistent with a 5 MW setpoint (exact field depends on your model)
- Telemetry for `ASSET_B` does **not** show the same change caused by that dispatch

Common failures

- Both assets change:
  - Agent applied setpoint to all assets, ignoring `asset_id`
- Neither changes:
  - Headend didn't route the setpoint to the stream, or agent didn't recognize the message
- HTTP 200 but no effect:
  - Dispatch accepted by REST but not delivered/applied (lack of ack is normal in demo-ready; check logs)

A6 (Optional): Site dispatch split by `site_id` (only if implemented)

> Skip this section unless you have implemented "dispatch by `site_id`" and a split policy.

Goal

A dispatch targeting a `site_id` is split deterministically into per-asset allocations and applied.

This plan assumes the split weight is **`capacity_mwhr`** from `assets.yaml`.

With the recommended `assets_test.yaml` values:

- `ASSET_A capacity_mwhr = 120`
- `ASSET_B capacity_mwhr = 40`

The weight ratio is **3:1**, so a 12 MW site dispatch should split to:

- `ASSET_A = 9 MW`
- `ASSET_B = 3 MW`

Actions

1. Issue a site dispatch (12 MW makes the split obvious):

   ```
   curl -s -X POST $HEADEND/dispatch \
     -H 'content-type: application/json' \
     -d "{\"site_id\":\"$SITE_ID\",\"mw\":12}" | jq .
   ```

2. Check gateway agent logs for the computed split (recommended log contents):

   - total requested MW
   - list of assets considered
   - each asset’s `capacity_mwhr`
   - computed `mw_i` before and after clamping

3. Verify with telemetry:

   ```
   sleep 10
   echo "After site dispatch (A):"
   curl -s $HEADEND/telemetry/$ASSET_A | jq .

   echo "After site dispatch (B):"
   curl -s $HEADEND/telemetry/$ASSET_B | jq .
   ```

Expected results

- Telemetry shows `ASSET_A` and `ASSET_B` moving toward their allocated setpoints.
- The split is stable between runs (same input → same allocations).
- The sum of per-asset allocations equals the requested total within rounding tolerance.

Success criteria (demo-ready)

- One gateway agent stays connected (no flapping) for at least 5 minutes
- `/telemetry/{asset_id}` returns distinct latest telemetry for at least 2 assets
- Per-asset dispatch changes only the targeted asset’s telemetry/behavior

---

Troubleshooting checklist

If `/dispatch` returns success but nothing changes

- Check headend logs: did it route to a connected stream for that `asset_id`?
- Check agent logs: did it receive a Setpoint? Did it match the `asset_id`?

- Confirm the gateway Register included that `asset_id` (assets list)

If telemetry only shows up for one asset

- Confirm agent is sending telemetry tagged with `asset_id` for each asset
- Confirm headend stores latest telemetry keyed by `asset_id`

If gRPC connection drops

- Add keepalive/idle timeout tuning (this is part of V1 pilot hardening, not demo-ready)