Endpoint Fast Path (EDS-Only Pushes)

Design for #9777.

1. Overview

In large clusters, endpoint updates are delayed by however long a full translation takes, because endpoints only reach Envoy by riding along in a complete rebuild of everything. Every EndpointSlice event takes the full pipeline today:

1. **Provider**: a full reconcile rebuilds the entire resource tree for the GatewayClass.
2. **Gateway API runner**: a full translation of all resources to IR.
3. **xDS runner**: a full translation of the IR to all xDS resource types, then a new snapshot.

While a build is in flight, incoming updates are merged into the next one, so endpoints continuously lag by roughly one full build duration. At large scale (tens of thousands of routes, expensive EnvoyPatchPolicy JSON patches), a single full translation can take multiple seconds to tens of seconds; under sustained pod churn, builds run back-to-back and endpoint staleness is sustained rather than episodic.

The structural mismatch: **endpoints change at pod timescale (rollouts, autoscaling, node drains) while routes and policies change at human timescale** — endpoint updates are orders of magnitude more frequent than any other input, yet they pay the cost of the most expensive machinery. Optimizing translation shrinks the staleness window; only decoupling removes the failure class where one tenant's expensive or invalid configuration delays endpoint delivery for every tenant sharing the control plane.

This document describes two options:

- **Option 1** adds an endpoint fast path *alongside* today's pipeline. Endpoint changes are also sent straight to the xDS layer, which republishes only the affected load assignments. The full pipeline is untouched and still runs for every event.
- **Option 2** separates configuration and endpoints into two independent paths in the provider, so ordinary endpoint churn stops triggering a full rebuild at all.

Option 1 requires fewer changes to the current architecture and is safer to land, while Option 2 provides greater savings. The recommendation is to ship Option 1 behind a flag and treat Option 2 as the follow-up.

Prior art: Istio's incremental (EDS-only) push — endpoint updates bypass `PushContext` recomputation, regenerate only affected `ClusterLoadAssignment`s, and escalate to a full

push when cluster-level config is affected (`DiscoveryServer.EDSUpdate`, `EndpointIndex.UpdateServiceEndpoints`).

2. Goals

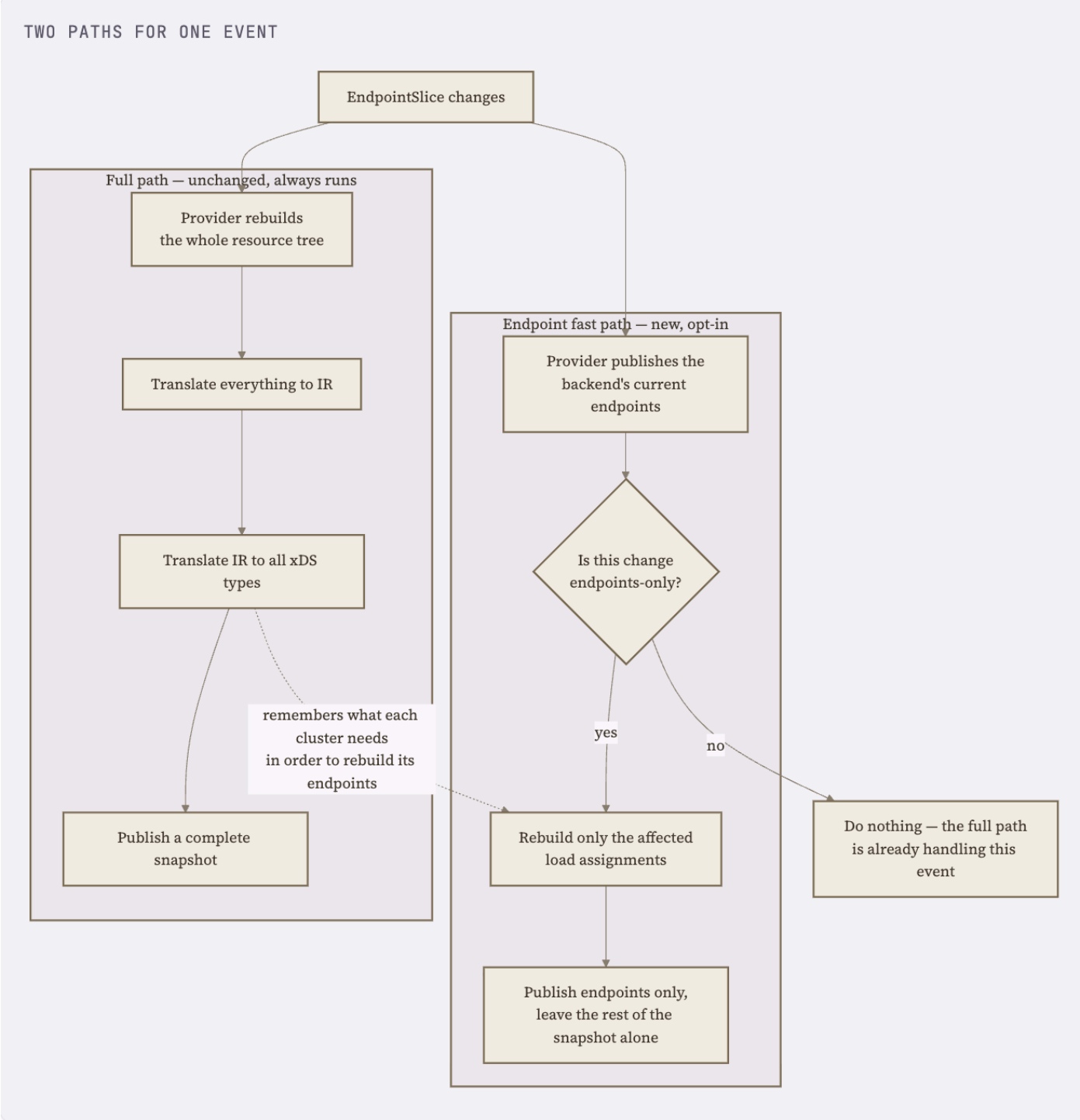
- Bound endpoint staleness independently of full-translation cost.
- Opt-in, with zero behavior change when disabled.
- Full builds remain authoritative; the fast path never produces state a full build would not eventually produce.
- Reuse the existing translation code on the fast path so both paths compute endpoints identically.

3. Non Goals

- Incremental translation of anything other than endpoints (no incremental CDS/LDS/RDS).
- Changing status reporting. Endpoint-derived status — a route's `BackendsAvailable` condition, and policies whose external-service backend has no endpoints — turns only on whether a backend has any endpoints, not on which ones. The fast path escalates that zero-crossing, so nothing it handles affects status.
- Covering DNS or dynamic-resolver backends. They never produce EDS resources, so there is nothing to fast-path.
- Changes to the xDS protocol usage (delta ADS remains as is).

4. Option 1: Endpoint fast path alongside the full path

4.1 How it flows



Two things happen when an EndpointSlice changes. The full path runs exactly as it does today. Alongside it, the provider also sends the affected backend's current endpoints straight to the xDS layer, which rebuilds only the load assignments of the clusters that backend feeds and publishes those alone.

The fast path never blocks or replaces the full path. When it cannot handle a change it does nothing at all, because the full reconcile for that same event is already running. That is what keeps this option safe: the worst a fast-path mistake can do is fail to make endpoints fresher.

4.2 Endpoints get their own channel

The provider gains a second handler on the EndpointSlice watch it already has, using the same filters, so the fast path only ever sees slices that would have triggered a reconcile anyway. For each event it works out which backend the slice belongs to, reads that backend's complete current set of slices, and publishes it on a new channel keyed by backend.

Publishing the whole set, rather than just the slice that changed, keeps the consumer simple: every message is self-contained, so when several pile up the newest one alone is enough.

4.3 The per-cluster endpoint context

Endpoint addresses alone are not enough to rebuild a load assignment. It also needs the backend's weights and priorities, its health-check overrides, and the zone-related settings — all of which come from configuration, not from endpoints. The destination settings must also stay in their original order, because some generated metadata refers to them by position.

The xDS translator records it as a by-product of each full translation. For every endpoint-based cluster it emits, it keeps a copy of exactly the inputs the load-assignment builder consumed:

```
// EndpointContext is what one EDS cluster needs in order to have its load
// assignment rebuilt outside a full translation: exactly the inputs the
// load-assignment builder consumed the last time this cluster was translated.
type EndpointContext struct {
   ClusterName string

   // Settings in their original order — position matters, because the
   // per-endpoint TLS transport-socket-match metadata embeds the index.
   // Each setting also records which backend and which service port its
   // endpoints came from.
   Settings []*ir.DestinationSetting

   // Health-check overrides that apply to individual endpoints.
   HealthCheck *ir.HealthCheck

   // Locality settings, from BackendTrafficPolicy.
   PreferLocal   *ir.PreferLocalZone
```

```
    WeightedZones []ir.WeightedZoneConfig
}
```

Contexts are held per gateway and indexed by backend, so an endpoint change for one Service resolves directly to the clusters that Service feeds. The same record carries the verdict on whether this gateway may use the fast path at all:

```
type gatewayContexts struct {
    // false when the gateway is excluded entirely, because an extension hook
    // may rewrite endpoints or clusters.
    enabled bool

    // backend ("<kind>/<namespace>/<name>") → the clusters it feeds
    byBackend map[string][]*EndpointContext
}
```

The fast path calls the same load-assignment builder with the same inputs and only the addresses changed, so both paths produce identical output by construction. Because the copy is taken from the pass that produced the live snapshot, what the fast path remembers and what Envoy is running cannot disagree.

4.4 Publishing endpoints without republishing everything

An xDS snapshot versions each resource type separately, but a full build stamps the same version on all of them. The fast path instead takes the snapshot that is currently live, copies it, replaces just the load assignments, and gives only the endpoint type a new version. Everything else keeps the version Envoy already has, so listeners, clusters and routes are not resent.

4.5 EnvoyPatchPolicy patches that target endpoints

An EnvoyPatchPolicy can patch a load assignment directly, so regenerating one would silently drop the patch. The fast path re-applies the patches that apply.

Patch application is already isolated per resource: each patch names a single resource, which is marshalled, patched, validated and written back on its own. So after rebuilding a load assignment the fast path applies exactly the patches naming it, in the same policy-then-patch order the full translation uses. The input is identical by construction — same builder, same remembered context, only the addresses differ — so the output is too.

If any of those patches fail to apply or fail validation, the fast path publishes nothing and leaves the update to the full path. The full path hits the same failure and records it on the policy's status, so a failure on the fast path needs no reporting of its own.

4.6 When the fast path steps aside

The fast path only handles changes that affect nothing but load assignments. Two kinds of condition hand a change back to the full path.

A gateway is excluded entirely — decided once per full build — when an extension server is registered for a hook that can modify endpoints or clusters. Extensions rewrite resources in place, so the fast path cannot tell what they changed.

Individual updates are skipped when:

- the backend is unknown — nothing references it yet, or no full build has happened;
- the endpoint count crosses zero in either direction, which changes much more than endpoints: clusters can appear or disappear, routes flip to direct 503 responses, and route status changes;
- a patch for one of the affected load assignments fails to apply, as described in 4.5.

What remains is the common case by a wide margin: addresses coming and going while the count stays above zero, readiness and draining flips, and zone changes.

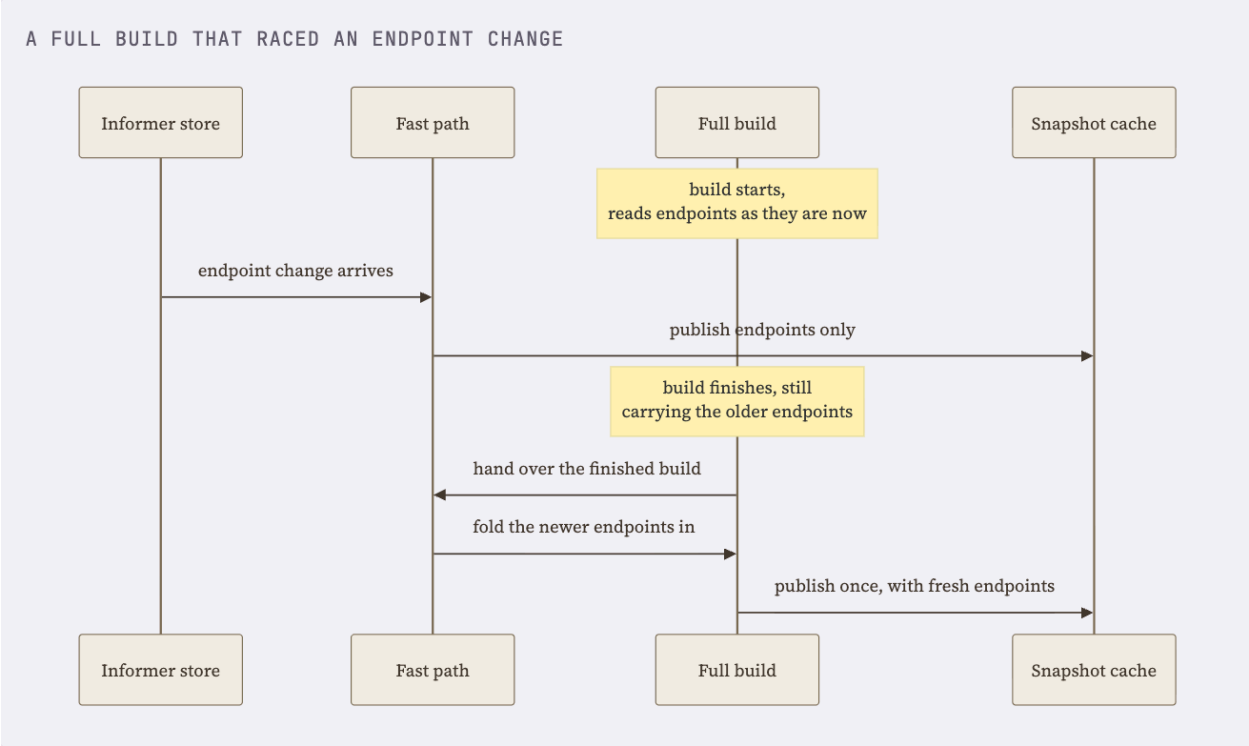
4.7 Keeping the two paths in agreement

The fast path makes a limited promise: **endpoints are fresh, and they are applied to the last configuration that translated successfully**. It does not promise the configuration itself is current — that is the full build's job.

Newer endpoints win. A full build reads endpoints when it starts, but takes a while to finish. If an endpoint changes in between, the fast path has already published the new value, and the finished build would put the old one back — sending traffic to a pod that has gone away. So before the build is published, any endpoint state newer than what it read is written into it. Envoy then gets a single push carrying the build's configuration and the latest endpoints, instead of a stale one followed by a correction.

There’s still a race condition where the full translation may not see the latest endpoints because the fast build hasn’t generated them yet. The fast build can finish first, only to have its update overwritten by the full translation later.

However, this is recoverable: the full translation will process subsequent update events and eventually publish the latest endpoints.



4.8 Turning it on

A runtime flag, `EndpointFastPath`, off by default, following the existing convention for behaviour that is new and worth guarding. It also requires the EndpointSlice index flag (on by default), which the provider uses to find a backend's slices.

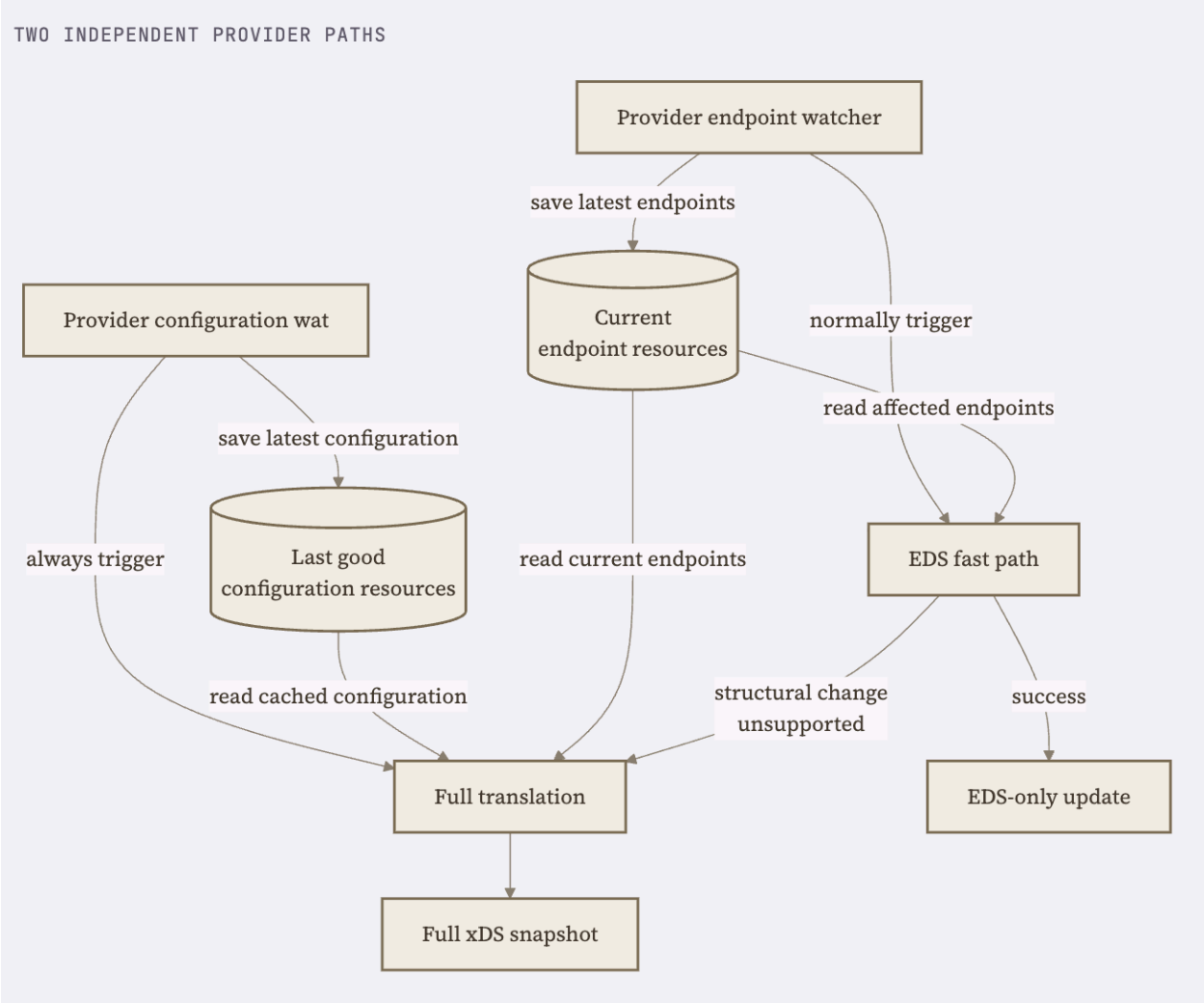
Two counters describe what the fast path is doing — patches applied, and updates skipped by reason — and the existing snapshot-update counter still counts pushes. The number to measure is how long an endpoint change takes to reach Envoy, flag off versus on, against a deliberately slow configuration.

5. Option 2: Separate configuration and endpoint paths

(Note: The Option 2 design is still an initial draft and needs more detail. It requires larger architectural changes and further discussion before we decide to move in this direction.)

Option 1 is additive, and that is both its safety and its limit: an endpoint event does *both* things — the fast path patches endpoints immediately, and the full pipeline still rebuilds the resource tree and re-translates everything. Staleness is bounded; total work is not reduced.

Removing that duplication requires moving **configuration resources and endpoint resources onto two independent event paths, each with its own cache and revision**.



5.1 Configuration updates

The configuration watcher handles changes to resources that can affect xDS structure: Gateways, Routes, policies, extension configuration, and the configuration-shaped parts of Services. On a configuration event the provider stores the latest valid configuration snapshot, then runs a full translation that combines it with the current endpoint resources and publishes a complete xDS snapshot.

Storing that snapshot is what makes the endpoint path self-sufficient: a fallback can run a full translation from the last-good configuration without waiting for another provider reconcile.

5.2 Endpoint updates

The endpoint watcher handles EndpointSlice changes independently. On an endpoint event the provider resolves the affected backend, reads its complete current slice set, updates the endpoint cache and its revision, and hands the update to the fast path, which regenerates only

the affected load assignments from what the last full translation remembered.

5.3 Full-translation fallback

An endpoint update falls back to full translation only for conditions decidable from the endpoint update plus what was remembered:

- nothing is remembered for the backend — which also covers a genuinely new backend, since a backend becomes referenced through a *configuration* change, which already triggers a full translation;
- the endpoint count crosses between zero and non-zero;
- the gateway is excluded because an endpoint-touching extension hook may rewrite load assignments, or a patch for one of the affected load assignments fails to apply, as described in 4.5..

Fallback combines the last-good configuration with the latest endpoints and runs the existing full translation, without another Kubernetes reconcile.

5.4 Compatibility surface

Splitting endpoints out of the configuration tree changes `resource.Resources`, whose `EndpointSlices` field is read by `egctl`, the file provider, the admin console config dump, and the offline translate path. Those consumers need to move to the endpoint cache or to a combined view.

6. Comparing the options

	Option 1 — fast path alongside	**Option 2 — separate paths**
Effort	Contained: one new channel, an IR field, remembered per-cluster inputs, an endpoint-only snapshot patch, and the escalation rules. No existing pipeline changes shape.	Large: a second provider pipeline, the resource tree split into two caches with independent revisions, a second full-snapshot producer, and migration of every consumer that reads endpoints from the tree.
Risk	Low. Purely additive and flag-gated; the unconditional reconcile remains the	Higher. Removing the unconditional reconcile makes fast-path correctness

	Option 1 — fast path alongside	Option 2 — separate paths
	backstop, so a fast-path mistake costs freshness, not correctness. Off by default.	load-bearing rather than best-effort, and two publishers double the ordering surface.
Memory	Adds the remembered per-cluster inputs, one retained slice set per referenced backend, and eagerly computed resource hashes. Endpoints are still also held in the full resource tree — resident twice.	Roughly neutral to better: endpoints live in one cache instead of being copied into every rebuilt tree, at the cost of a durable last-good configuration snapshot.
CPU	Per endpoint event: *adds* the fast-path work on top of the unchanged full reconcile and translation. Saves the IR→xDS translation, validation and full rehash only on the push the fast path serves.	Per endpoint event: removes the full reconcile and full translation entirely for the common case. This is where the actual CPU reduction lives, and it grows with configuration size.
Delivers	Bounded endpoint staleness, independent of configuration cost.	Bounded staleness *and* endpoint churn that no longer costs a full rebuild.

6.1 Rough magnitudes

There are no measurements yet. The figures below are modelled from the structure of each option against one example deployment, to show which way each cost moves and roughly how far; the propagation-latency benchmark is what should settle them.

The example: a churn-dominated deployment — roughly 10,000 routes, 2,000 endpoint-based clusters, heavy patch use. A full translation takes about 5 s, and about 95% of events are endpoint changes.

	Option 1	Option 2
CPU	≈ +0.2%	≈ −95%

	Option 1	Option 2
Memory	≈ +20–30%	≈ −10–20%

CPU. Option 1 adds the fast path on top of an unchanged full build, so its extra cost is rebuilding the affected load assignments plus a snapshot clone — a few milliseconds against a 5 s translation. Option 2 removes the full translation for endpoint-only events, so 95 events in 100 drop from ~5 s to ~10 ms: `(95×10ms + 5×5s) / (100×5s)`.

Memory. Option 1 keeps roughly two further copies of each referenced backend's EndpointSlices — one retained on the channel, one held as the last-seen update — plus a copy of the endpoint-derived settings per cluster, so the overhead tracks how much of the heap endpoints already occupy. Option 2 stops copying endpoints into every rebuilt tree, recovering part of that.

The shape to take away: Option 1 costs a little to remove the wait, and Option 2 is where the work itself goes away.

7. Recommendation

Ship Option 1 behind the flag, and treat Option 2 as the next step if we need to further reduce control-plane CPU and memory usage.

Option 1 is small, reversible, and safe by construction. It changes nothing about how configuration reaches Envoy, and the full reconcile that still runs on every endpoint event is exactly what makes its best-effort behaviour acceptable. It delivers what the issue asks for — endpoint staleness that no longer scales with configuration cost — without making anything else depend on the new code being right.

What it does not do is reduce work. Every endpoint event still pays for a full rebuild; the fast path is simply quicker at getting endpoints out. Option 2 is where that cost disappears, because endpoints stop being part of the tree that gets rebuilt.

Option 2 removes the unconditional reconcile and significantly reduces the control-plane work.

Suggested order:

1. Option 1 behind `EndpointFastPath`, default off, with the propagation-latency benchmark.
2. Enable by default once the skip reasons and latency data look right.
3. If control-plane CPU or memory is still a concern, Option 2: split the provider's caches, stop reconciling the world for endpoint-only events.

8. Alternatives considered

- **Optimizing translation only.** Worthwhile independently, but leaves endpoint freshness coupled to worst-case configuration cost; the failure class remains.
- **A side-channel context artifact published next to the IR**, or the provider shipping Service port information with each event. Both split what the fast path needs across sources that must be kept in sync; deriving it from the IR plus what the translation pass remembered keeps a single source of truth.
- **Re-running extension endpoint hooks or CLA patches on the fast path.** Would keep the fast path available for extension users, but puts gRPC hook calls on the hot path and duplicates patch-application semantics; deferred until there is demand.