Graph Agent Protocol (GAP): Unifying Typed Discovery, Batched Federation, and Zero-Trust Governance for AI Agents

Duy Đức Nguyễn · Infinite Agent Labs · `office@infiniteagentlabs.com` · ORCID: 0009-0009-4741-5637

Reference Release `v0.1.0-rc.1` · November 16, 2026 · Apache-2.0 · Code & Artifacts: https://github.com/vnknowledge2014/GraphProtocol

Abstract. AI agents increasingly invoke REST/OpenAPI APIs, SQL stores, GraphQL subgraphs, gRPC endpoints, CLI utilities, codebases, Model Context Protocol (MCP) tool servers, peer Agent2Agent (A2A) workers, and interactive AG-UI interfaces. Current deployments fragment this plane across flat JSON-RPC catalogs (MCP), opaque peer task endpoints (A2A), and ad-hoc frontend streams, causing $O(N)$ prompt schema inflation (**203,421 tokens** at N=1,000), client-orchestrated N+1 join round-trips, unpinned descriptions vulnerable to rug pulls, and hot-path latency from neural routing. We introduce **Graph Agent Protocol (GAP)**, a strongly typed GraphQL-profile gateway built across three separated planes: (1) a *Zero-ML deterministic execution plane* with Cedar policy checks, tenant-keyed planning, batched DataLoader joins, and wait-free immutable snapshots backed by a pure-Rust persistent AVL `OrdMap`; (2) a *three-level progressive discovery plane* unifying MCP `2026-07-28` (**651 tokens** constant), A2A v1.0, and AG-UI v1.0 (`/ag-ui` SSE + interrupts) alongside multi-provider System-1 routers (`@receptron/laya`, `Jev`, `gemini-decision-1`) and automatic deterministic fallback (`INV-17`); and (3) a *zero-trust control plane* enforcing propose-not-activate ingestion, SHA-256 description-hash drift freezes, DNS-pinned SSRF egress guards, OS sandboxing, and BLAKE3 hash chains over dual-stack `HTTP/2` + `HTTP/3` (`QUIC RFC 9114`, 0-RTT disabled). Implemented across 32 layered Rust crates with zero `RUSTSEC` advisories, GAP cuts cross-branch token usage at N=1,000 by **99.0%** uncached (**93.5%** cached) and code-graph context by **119.3x–208.7x**, boots its 28.1 MB bit-identical binary in **1055.0 ms** at **9.32 MiB** idle RSS, sustains **1,063.6 req/s** over TCP (**11.54 ms** p95) and **5,013.1 req/s** over persistent multiplexed QUIC (**2.36 ms** p95) with zero errors, and recovers from OS `SIGKILL` in **28.86 ms**.

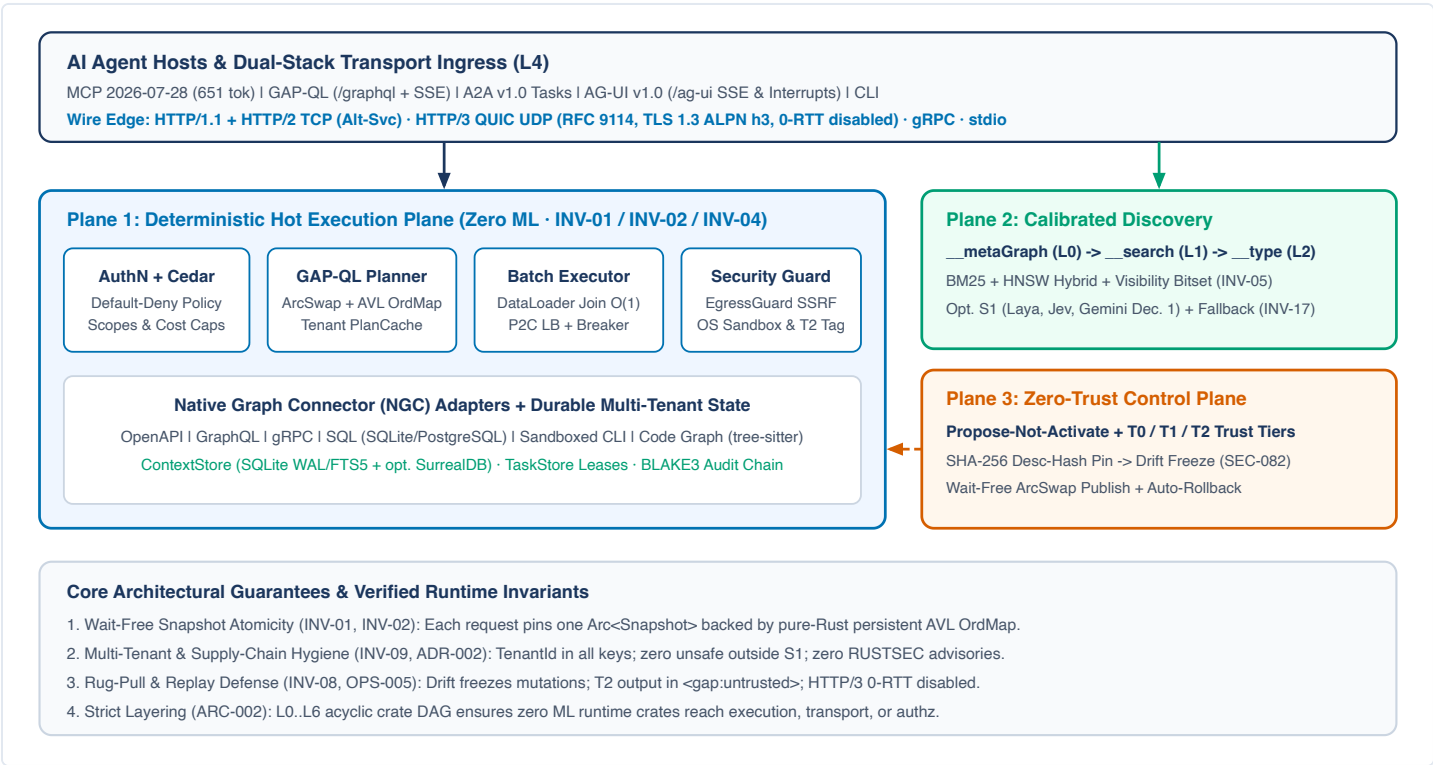


Figure 1: Three-plane zero-trust architecture of Graph Agent Protocol (GAP). Plane 1 runs typed GraphQL planning, Cedar authorization, and batched joins with zero ML models (`ARC-002`, `INV-01`, `INV-04`) behind dual-stack HTTP/1.1+HTTP/2 TCP and HTTP/3 QUIC transport (`OPS-005`). Plane 2 performs visibility-filtered progressive disclosure; Plane 3 governs proposal verification, rug-pull drift pinning, and lock-free snapshot swaps.

1. Introduction & Threat Model

Production AI agents must discover, authorize, compose, and audit operations across heterogeneous backends. Anthropic's Model Context Protocol (MCP) [1] standardizes vertical tool invocation over JSON-RPC, whereas Google's Agent2Agent (A2A) protocol [2] standardizes horizontal multi-agent delegation via Agent Cards and asynchronous task lifecycles. Operating multi-tenant gateways exposes four structural weaknesses that flat catalogs and point-to-point RPC do not solve together:

1. **Prompt Context Explosion & N+1 Waterfalls:** Flat MCP exports grow linearly to **203,421 tokens** at N=1,000 before reasoning begins, forcing models to issue sequential multi-turn round-trips to join cross-backend entities.

2. **Semantic Rug Pulls & Spoofed Hints:** Security audits [3,4,5] show unpinned tool descriptors permit hidden prompt injection and post-approval behavioral shifts.
3. **Hot-Path Neural Entanglement:** Invoking ML tool classifiers on every request adds tens of milliseconds and mixes probabilistic inference with security-critical authorization.
4. **Protocol Silos Without Typed Federation:** Treating SQL tables, REST APIs, CLI binaries, and code ASTs as unrelated untyped tools forces the LLM to act as query planner and join engine.

GAP enforces a simple engineering rule: use deterministic compilers, typed graphs, and formal policy engines wherever structure exists, and confine probabilistic retrieval strictly to bounded, calibrated intent discovery (`ARC-002`).

2. Three-Plane Architecture & Invariants

As Figure 1 shows, GAP assigns every downstream backend (*branch*) an explicit trust tier: **`T0`** (Ed25519-signed internal manifest; signature mismatch downgrades fail-closed to `T2`), **`T1`** (admin-approved and pinned via cryptographic `spec_hash` and `desc_hash`), and **`T2`** (untrusted external branches whose text outputs are length-capped, scrubbed, and wrapped inside escaped `<gap:untrusted>` envelopes, with CLI/stdio blocked without OS sandboxing).

To prevent architectural erosion, 20 normative invariants (`INV-01`..`INV-20`) are enforced continuously via static crate layering checks (`tools/check_layers.py`) and strict spec traceability (`tools/spec_trace.py --strict`):

- **`INV-01` / `INV-02` (Pinning & Lock-Free Hot Path):** Requests pin one `Arc<Snapshot>` on ingress; read paths forbid `Mutex`, `RwLock`, and `DashMap`, using wait-free `ArcSwap` and persistent AVL `OrdMap` trees.
- **`INV-04` / `INV-05` (Deterministic Authz & Visibility):** Compiled Cedar rules [10] evaluate before execution and never consume ML scores; `__search` applies per-principal visibility bitsets before ranking.
- **`INV-08` / `INV-09` (Pinned Descriptors & Tenants):** Planners read sanitized pinned descriptions only; caches, quotas, stores, and BLAKE3 audits include `TenantId`.
- **`INV-17` / `INV-18` / `INV-20`:** System-1 ML cannot block typed execution; remote joins execute in bounded batched frames; peer A2A delegation tracks hop counters to reject cycles.

3. Protocol Design & Dual-Stack Transport

3.1 GAP-QL & Three-Level Progressive Disclosure

GAP-QL is a strict GraphQL profile parsed by `apollo-compiler` with zero custom grammar keywords [9]. Capabilities—`__metaGraph` (L0 branch summaries), `__search` (L1 hybrid BM25+HNSW candidates), `__type` (L2 exact SDL hydration), `@sideEffect`, `@requiresScope`, and `@join`—live purely as standard types and directives. Behind MCP `2026-07-28`, the five-tool facade (`gap_search`, `gap_describe`, `gap_query`, `gap_mutate`, `gap_task`) holds `tools/list` invariant at **651 tokens**.

3.2 Three-Axis Facades, Native Connectors & Dual-Stack HTTP/2 + HTTP/3

Six reference Native Graph Connectors (`OpenAPI`, `GraphQL`, `gRPC`, `SQL`, sandboxed `CLI`, `tree-sitter Code`) translate heterogeneous protocols into `Describe`, `BatchQuery`, and `Mutate` frames. At L4, `gap-transport` serves `/graphql`, `/graphql/stream`, `/mcp` (MCP `2026-07-28`), A2A v1.0, and `/ag-ui` (AG-UI v1.0 SSE + human approval interrupts, spotlighted via `<gap:untrusted>`, `ADR-027`) over **HTTP/1.1+HTTP/2 TCP** (`Alt-Svc: h3`) and **HTTP/3 QUIC** (`RFC 9114`, `0-RTT` disabled, `OPS-005`). Optional System-1 routing (`ADR-028`) unifies Candle, `@receptron/laya`, `Jev`, and `gemini-decision-1` (0 output tokens, up to 1,000 parallel heads) under `INV-17` baseline fallback.

4. Zero-Trust Governance & Snapshot Control

All changes enter via sandboxed `gap-ingest` extraction, CISG injection linting, and admin-approved `Proposal` records rather than auto-mutating live routing. `DriftMonitor` hashes sanitized upstream descriptions and immediately freezes tampered branches to `BranchStatus::Drift` while planners continue serving pinned baselines (`SEC-082`).

Outbound HTTP passes through `EgressGuard`, which pins resolved DNS sockets once, blocks loopback/RFC 1918/metadata IPs (`169.254.169.254`) and obfuscated numeric/IPv6-mapped forms across redirects. Subprocesses run inside Linux Bubblewrap or macOS Seatbelt sandboxes with SHA-256 binary allowlists. Snapshots publish off-thread via `SnapshotHandle` with automatic 60 s error-spike rollback and BLAKE3 chained audit logs.

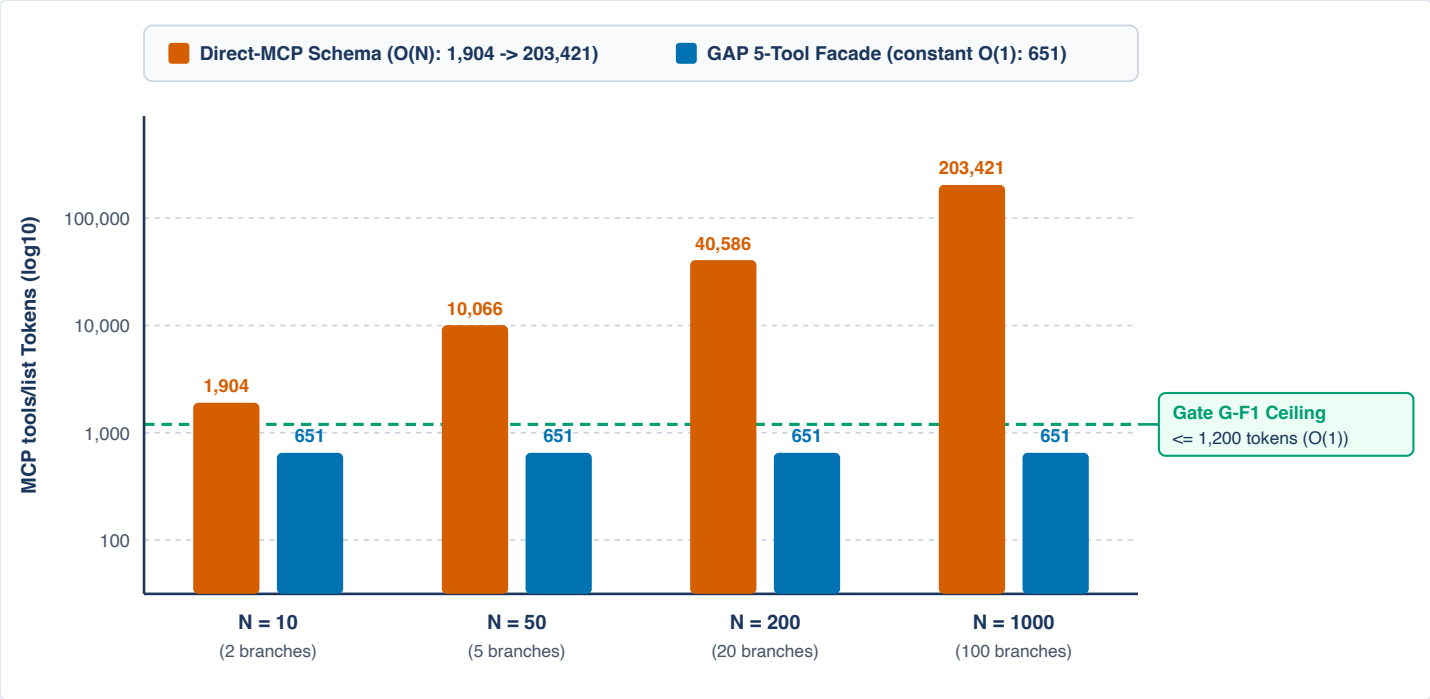


Figure 2: Static MCP `tools/list` token scaling (`GQL-022` estimator). Publishing flat tool catalogs explodes to 203,421 tokens at N=1,000 tools across 100 branches, whereas GAP's five-tool progressive facade remains fixed at **651 tokens** (46% below Gate G-F1's 1,200-token ceiling).

Table 1: Reproducible `GAP-Bench` end-to-end comparison across catalog scales (N), uncached vs. 0.10x warm-prefix cached regimes, and agent turn counts (100% task success across all cells; ratio < 1.000x denotes GAP token savings).

Suite	N	Direct Unc.	GAP Unc.	Ratio (U)	Direct Cac.	GAP Cac.	Ratio (C)	Turns D → G
api_lookup	10	2,041.6	5,823.0	2.852x	547.2	4,691.9	8.575x	1.0 → 2.0
cross_branch_join	10	8,891.8	8,517.9	**0.958x**	2,200.5	7,383.7	3.355x	4.0 → 2.0
api_lookup	50	10,205.5	5,859.5	**0.574x**	2,385.2	4,728.0	1.982x	1.0 → 2.0
cross_branch_join	50	41,539.8	8,547.9	**0.206x**	6,740.7	7,413.7	1.100x	4.0 → 2.0
api_lookup	200	40,724.3	5,865.5	**0.144x**	9,251.2	4,734.9	**0.512x**	1.0 → 2.0
cross_branch_join	200	163,619.8	8,547.9	**0.052x**	23,717.4	7,413.7	**0.313x**	4.0 → 2.0
api_lookup	1000	203,559.5	5,878.0	**0.029x**	45,889.1	4,747.0	**0.103x**	1.0 → 2.0
cross_branch_join	1000	814,959.8	8,547.9	**0.010x**	114,294.4	7,413.7	**0.065x**	4.0 → 2.0
code_graph	24	141,681.3	1,187.4	**0.008x**	137,810.4	660.4	**0.005x**	11.0 → 1.0

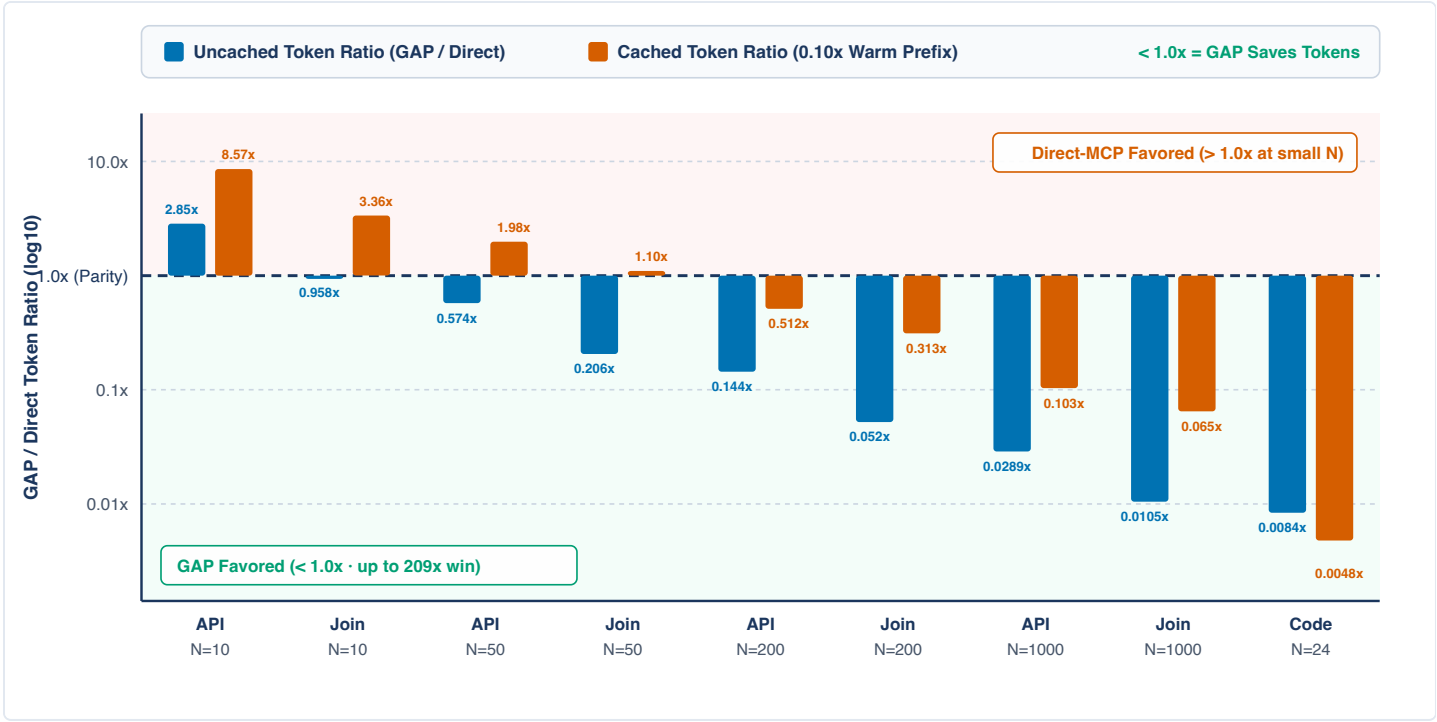


Figure 3: Crossover regime law plotting `GAP / Direct-MCP` prompt token ratio on a log scale. Below 1.0x (mint region), batched federation and progressive disclosure cut tokens by **95.3x–208.7x**; above 1.0x (rose region), tiny cached catalogs ($N \leq 50$) still favor Direct-MCP.

5. Rust Implementation & Formal Traceability

GAP is built in Rust Edition 2024 (`rustc 1.98.1`) across 32 crates (`L0`–`L6`, including `gap-agui` and multi-provider `gap-s1`), forbidding `unsafe` everywhere outside optional `gap-s1` and denying `unwrap`, `expect`, and `panic` in non-test paths. BSL dependencies remain strictly behind optional `store-surreal`. To eliminate every unmaintained/unsound third-party crate from the supply chain, snapshot maps use an owned zero-unsafe AVL `OrdMap` and PEM parsing uses `rustls-pki-types` directly, achieving **0 errors** across `cargo deny check advisories bans licenses sources` (`ignore = []`).

Verification passes end-to-end across **362 specification requirements** (89 `MUST`, 0 uncovered across 2,515 markers), **135/136** offline/loopback conformance checks, **365** 1,024-case proptests, 10 fuzz targets (34 corpora), chaos, soak, and upgrade/backup drills.

6. Experimental Evaluation & Crossover Analysis

Across Table 1 and Figure 3, `GAP-Bench` demonstrates both massive wins and honest boundaries (`ADR-017`, `ADR-021`): at N=1,000, GAP cuts cross-branch join tokens by **99.0%** uncached (**93.5%** cached) and halves turns (4.0 → 2.0); on `code_graph` (24 symbols), typed graph walks reduce turns from 11.0 to 1.0 for **119.32x** uncached and **208.68x** cached savings. Conversely, on tiny flat catalogs (N=10 cached: 547.2 vs. 4,691.9 tokens), single-turn Direct-MCP stays smaller, and conditional `X-1` wire columnarization yielded <20% suite reduction and remains deferred.

Table 2 isolates `RouteBench` discovery routing: un-finetuned Candle weights score only 14.29% debiased top-1 at 73.85 ms p95, whereas GAP's deterministic lexical+structural baseline achieves **95.24%** test / **97.22%** held-out top-1 in **0.887 ms**, justifying fail-safe `BASELINE_FALLBACK` under Gate `G-S1`.

Table 2: `RouteBench` comparison (n_{test}=48, n_{held}=72; 1,000 bootstrap resamples), lock-free micro-primitives (`m3-baseline.json`), and live bit-identical dual-stack `HTTP/2` + `HTTP/3 QUIC` verification drills (`bench/results/ga_local/report.json`).

Benchmark / Drill Component	Metric 1	Metric 2	Latency / Throughput	Artifact Source
`baseline (BM25+Struct)`	Test Top-1: **95.24%**	Held Top-1: **97.22%**	**0.887 ms** p95	`s1_comparison.json`
`s1_forward (laya-candle:f32)`	Test Top-1: 7.14%	ECE_{15}: 0.0445	42.34 ms p95	`s1_comparison.json`
`s1_debiased (Laya + PMI)`	Test Top-1: 14.29%	ECE_{15}: **0.0303**	73.85 ms p95	`s1_comparison.json`
`Lock-Free Balance / RCU`	ArcSwap load: **15 ns**	Guard / CAS / P2C: 42 / 44 / 94 ns	Store p99 @256r: 59 µs	`m3-baseline.json`
`gap serve (TCP HTTP/1.1+2)`	Cold start: **1055.0 ms**	Idle / Peak RSS: **9.32 / 16.30 MiB**	**1,063.6 req/s** (11.54 ms p95, 0 err)	`ga_local/report.json`
`HTTP/3 QUIC (8 mux conns)`	TLS 1.3 ALPN `h3`	0-RTT: **Disabled** · Alt-Svc	**5,013.1 req/s** (2.36 ms p95, 0 err)	`ga_local/report.json`
`Binary Parity & Crash`	Size: 28,084,064 B	Double SHA-256: **Identical**	SIGKILL recover: **28.86 ms**	`ga_local/report.json`

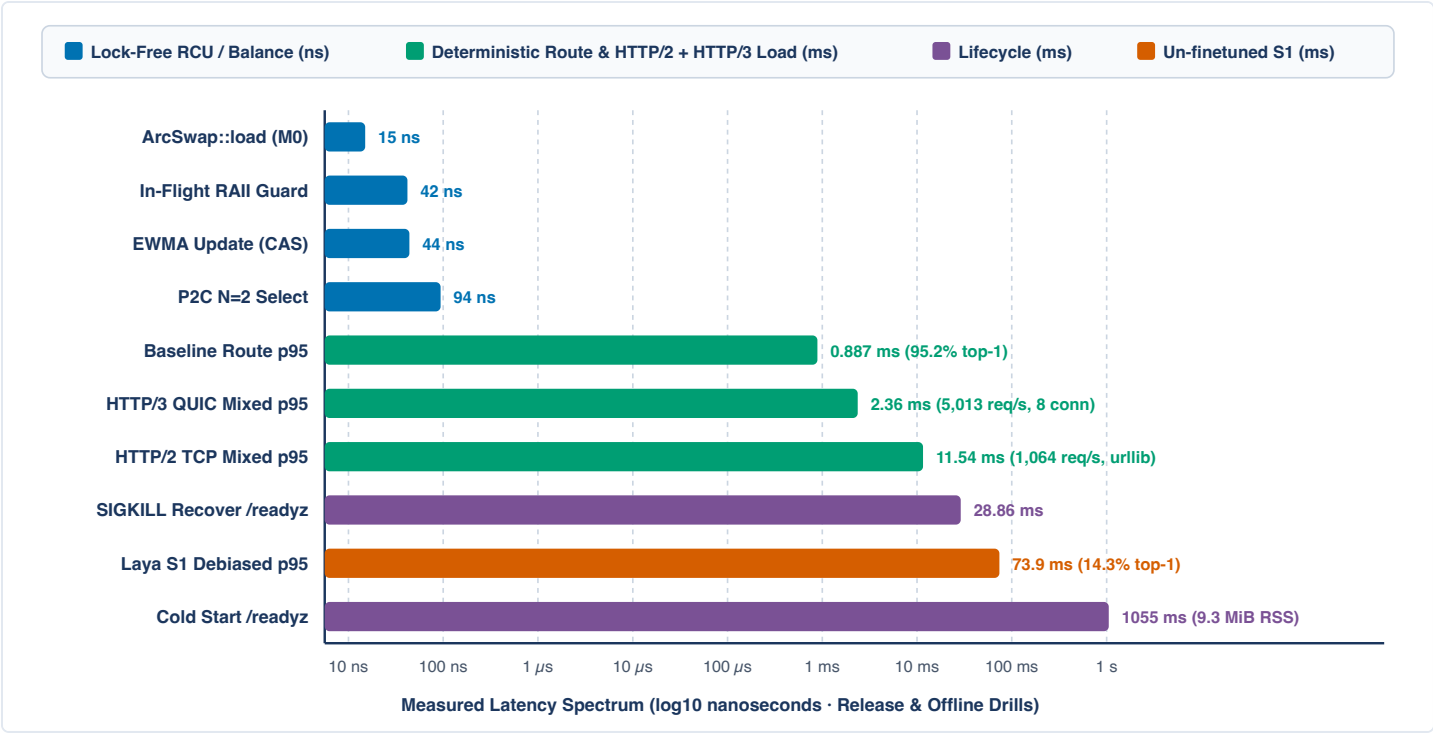


Figure 4: Multi-scale latency spectrum across lock-free RCU/balance primitives (15–94 ns), deterministic baseline discovery (0.887 ms), live multiplexed HTTP/3 QUIC (2.36 ms p95) and HTTP/2 TCP (11.54 ms p95) mixed loads, and OS `SIGKILL` recovery (28.86 ms).

7. Related Work & Limitations

Retriever-assisted tool routers such as Gorilla [7] and ToolLLM [8] expand API coverage yet omit deterministic Cedar authorization [10], conformal abstention [11], and cross-backend DataLoader joins. Recent MCP security audits [3,4,5] and A2A security analyses [6] catalog poisoning and rug pulls; GAP neutralizes these via propose-not-activate ingestion, `desc_hash` pinning, `EgressGuard`, and sandboxing. GraphQL MCP servers [12] pioneered progressive GraphQL discovery, which GAP extends across arbitrary non-GraphQL backends over HTTP/2 and QUIC.

Multi-day cloud soak runs (72 h), 24 h libFuzzer sweeps, live frontier LLM sweeps, and Linux OIDC signing remain CI/cloud release gates rather than local macOS drills; locally all HTTP/2 (short-lived Python `urllib` clients to subprocesses) and HTTP/3 (persistent QUIC streams to in-process fixtures) benchmarks measure distinct transport topologies reported transparently above.

8. Conclusion

GAP reconciles MCP usability, A2A horizontal tasks, dual-stack HTTP/2 and HTTP/3 QUIC performance, and enterprise zero-trust security in one verified Rust gateway. All specifications, conformance tests, reproducible benchmarks, and Apache-2.0 source artifacts are openly published for verification and reuse.

References

1. Anthropic. *Model Context Protocol (MCP) Specification* (incl. 2026-07-28 revision). `https://modelcontextprotocol.io`, 2024–2026.
2. Google LLC and A2A Contributors. *Agent2Agent (A2A) Protocol Specification*. `https://a2a-protocol.org`, 2025.
3. H. Song et al. Beyond the Protocol: Unveiling Attack Vectors in the Model Context Protocol Ecosystem. *arXiv:2506.02040*, 2025.
4. Z. Wang et al. MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers. *arXiv:2508.14925*, 2025.
5. arXiv:2512.06556 Authors. Semantic Attacks on Tool-Augmented LLMs: Securing MCP Against Descriptor Manipulation, 2025.
6. arXiv:2504.16902 Authors. Building a Secure Agentic AI Application Leveraging A2A Protocol, 2025.
7. S. G. Patil et al. Gorilla: Large Language Model Connected with Massive APIs. *arXiv:2305.15334* (NeurIPS), 2023.
8. Y. Qin et al. ToolLLM: Facilitating LLMs to Master 16000+ Real-world APIs. *arXiv:2307.16789* (ICLR), 2023.
9. O. Hartig and J. Pérez. Semantics and Complexity of GraphQL. In *Proc. WWW '18*, 2018.
10. J. W. Cutler et al. Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization. *Proc. ACM Program. Lang. (OOPSLA)*, 2024.
11. A. N. Angelopoulos and S. Bates. A Gentle Introduction to Conformal Prediction. *arXiv:2107.07511*, 2021.
12. Apollo GraphQL and Grafbase Engineering. Progressive Schema Disclosure Patterns in GraphQL MCP Servers, 2025.