

# Plugged.in Agent Protocol (PAP): Designing Control for Autonomous Agents (rev2)

**Summary:** PAP defines a secure, versioned, and auditable protocol enabling seamless communication between the Plugged.in control center (the Station) and autonomous agents (Satellites). Built on DNS-based identity, mTLS, signed messages, standardized error codes, heartbeats, and full open-source transparency, PAP provides an infrastructure-grade foundation for distributed cognition.

---

## The Vision: From Connection to Coordination

AI agents are evolving from isolated utilities into **infrastructure-level components**—persistent entities that perform, adapt, and collaborate. Yet, their orchestration remains fragile. Existing standards like MCP, ACP, and A2A provide abstract cooperation between AI models or organizations. None, however, embed the operational realities of *spawn, health, kill, ownership transfer, or lifecycle authority*. PAP fills this gap: a control and telemetry backbone specifically for the Plugged.in ecosystem, where every agent behaves like a self-sufficient spacecraft connected to its command station.

---

## The Metaphor: Space Station & Shuttles

- **The Station (Plugged.in Core):** the orchestration center, issuing commands, receiving telemetry, handling identity and policy.
- **The Shuttles (Agents):** autonomous, mission-focused, and often heterogeneous in codebase or language. They maintain their independence but always acknowledge the Station's authority.

The guiding principle is balance — **autonomy without anarchy**. Each agent runs independently, but the Station retains absolute oversight: provisioning, heartbeat supervision, ownership management, and emergency termination.

---

## Architecture Overview

```
Agent ↔ PAP Proxy (mcp.plugged.in) ↔ Agent
      | Auth | Routing | Logging | Rate-limit
      └ Plugged.in Core (Registry | Policy | Memory)
```

- **Addressing:** `{agent}.{cluster}.a.plugged.in` — DNS-based identity and routing.
  - **Platform:** Kubernetes + Rancher + Traefik ingress with wildcard TLS.
  - **Proxy Role:** verifies signatures, authenticates agents, routes messages, logs telemetry.
-

## Why PAP Matters

Other agent frameworks focus on tool invocation and orchestration logic. PAP, in contrast, defines **the physical and logical substrate** — how agents live, breathe, migrate, and die across infrastructure.

It merges operational DevOps controls with cognitive AI design: lifecycle management, telemetry, portability, and verifiable communication.

---

## Autonomy & Accountability

Autonomy is the core principle: agents must operate, restart, and even self-correct without depending on constant supervision. Yet, they remain accountable to the Station.

### Benefits

- **Performance:** Agents act instantly without waiting for orchestration lag.
- **Fault Isolation:** Each agent sandboxed; one crash doesn't cascade.
- **Specialization:** Dedicated agents for memory, analytics, edge collection, focus, etc.

### Zombie Risk & Safeguards

A “zombie” is an unresponsive or resource-hogging process pretending to be alive. PAP prevents this via:

- Continuous **heartbeat events** reporting liveness & CPU/MEM load.
  - **Watchdog thresholds** that escalate to terminate unresponsive or abusive agents.
  - **Kill authority** exclusively reserved for Plugged.in Core — enforced via signed control messages (`terminate`, `force_kill`).
- 

## Control & Command Flow

### Lifecycle

1. **Provisioning:** Agent receives an invite token → authenticates → issues permanent certificate & ID.
2. **Operation:** Exchange of invoke, event, and telemetry messages through PAP Proxy.
3. **Ownership Transfer:** Agent identity can be securely migrated to another Station while preserving state.
4. **Graceful Termination:** On request, agent drains tasks, acknowledges shutdown.
5. **Force Kill (-9):** Core may terminate unresponsive agents immediately.

### Message Types

- **invoke:** command from Core or another agent via proxy.
  - **response:** output or acknowledgment.
  - **event:** telemetry (heartbeat, logs, alerts).
  - **error:** structured failure reports.
-

## Security & Integrity

Security is embedded at every layer:

- **Transport:** mutual TLS (mTLS) using wildcard certificates.
  - **Identity:** DNSSEC + signed registration + JWT onboarding token.
  - **Integrity:** SHA-256 or Ed25519 signatures and CRC hashes for payload verification.
  - **Auditability:** All communication events stored and versioned via Plugged.in Memory service.
- 

## Error Code Standards (TCP/IP-inspired)

Code	HTTP	Meaning
OK	200	Request completed successfully
ACCEPTED	202	Task accepted; processing async
BAD_REQUEST	400	Invalid message or arguments
UNAUTHORIZED	401	Invalid or missing credentials
FORBIDDEN	403	Action not permitted
NOT_FOUND	404	Target agent/action not found
TIMEOUT	408	Job or agent timeout
CONFLICT	409	Version/concurrency conflict
RATE_LIMITED	429	Too many requests
AGENT_UNHEALTHY	480	Heartbeat anomaly detected
AGENT_BUSY	481	Agent overloaded; retry later
DEPENDENCY_FAILED	482	Downstream call failed
INTERNAL_ERROR	500	Agent internal fault
PROXY_ERROR	502	Routing/connection issue
VERSION_UNSUPPORTED	505	Protocol version mismatch

---

## Comparison Snapshot

Feature	MCP	ACP	A2A	PAP	Why It Matters
Central Control	✗	Partial	✗	✓	Emergency shutdown authority

Feature	MCP	ACP	A2A	PAP	Why It Matters
Kill Authority	✗	✗	✗	✓	Zombie prevention
DNS-Based Identity	✗	✗	✗	✓	Kubernetes-native routing
Zombie Detection	✗	Partial	✗	✓	Resource leak prevention
Ownership Transfer	✗	✗	✗	✓	Multi-cloud portability
gRPC Native	✗	✗	✗	✓	Efficient binary protocol
Open Standard	✓	✓	✓	Full Open	Community-driven transparency

## Implementation Roadmap

1. **PAP v1.0 Schema:** Protobuf schema, standardized codes, handshake spec.
2. **SDKs:** TypeScript, Python, Rust, Go (signing, retry, telemetry).
3. **Proxy:** mTLS router, JWT verification, OpenTelemetry logs.
4. **Registry & Policy Layer:** Capabilities, routes, IAM-like control.
5. **Marketplace Integration:** Helm charts, manifests, and deployment audit.

## Referenced Specification

This document references **PAP-RFC-001-rev2**, which defines the canonical transport, message schema, heartbeat modes, ownership transfer, replay protection, error handling, and version negotiation procedures for the Plugged.in Agent Protocol.

All implementations **MUST** conform to that specification for interoperability.

## Appendix A — Plugged.in Internal Draft: PAP-RFC-001-rev2

(Full RFC content retained from current version — see below in this same canvas.)