NodePulse: A Decentralized Node Registry Protocol for the Plexum Network

Version 1.0 — March 2026

Abstract

NodePulse is a decentralized peer-to-peer node registry protocol that enables personal devices — including smartphones and desktop computers — to operate as publicly reachable servers without static IP addresses, domain names, or cloud infrastructure. Nodes expose local HTTP services through relay tunnels (Cloudflare Tunnel, Tor Hidden Services, or VPS-based reverse proxies), announce their ephemeral URLs to the network with RSA-SHA256 cryptographic signatures, and propagate state via a gossip protocol. A browser-based recovery system ensures that visitors can always re-discover a node even when its tunnel URL changes, using Service Workers and cached seed lookups. The network that emerges from the protocol is called **Plexum**.

Table of Contents

---

1. Introduction

The modern internet concentrates web hosting behind a small number of cloud providers. Publishing a website or web application requires a domain name, a server with a static IP address, and ongoing infrastructure costs. For billions of users on mobile devices or behind NAT, operating a server is effectively

impossible.

NodePulse inverts this model. Any device with a PHP runtime and an internet connection can become a publicly reachable server. The device runs a local HTTP server (typically PHP-CGI behind nginx on port 8080), exposes it through a relay tunnel that assigns an ephemeral public URL, and announces that URL — cryptographically signed — to a decentralized registry. Other nodes on the network discover, verify, and propagate this announcement through gossip. When the tunnel URL changes (as it does on every restart with free Cloudflare tunnels), the node simply signs and announces the new URL. A browser-side recovery mechanism ensures that visitors are never permanently lost.

The result is a network where an Android phone in a drawer, a Windows laptop, or a Raspberry Pi can serve web applications to the public internet, with cryptographic identity that persists across URL changes, without requiring any centralized registration authority.

---

2. Design Goals

Minimal infrastructure. A node requires only bash, PHP, OpenSSL, and curl. No databases, no containers, no heavy runtimes. Installation is a single shell script.

Relay-agnostic. The protocol treats the public URL as an opaque string. The relay mechanism (Cloudflare Tunnel, Tor, frp, rathole) is an implementation detail. The cryptographic signature is the sole proof of identity — not the domain name.

Zero configuration n etworking. New nodes generate an identity, connect to hardcoded bootstrap seeds, announce themselves, and begin participating in gossip. No manual peer configuration is required.

Offline re silience. When a node's tunnel dies, visitors are served a cached Recovery Browser from the Service Worker. The Recovery Browser queries seed nodes to find the node's new URL and redirects the visitor automatically.

PHP compatibility. The entire backend runs on PHP 5.6 through 8.x, enabling deployment on constrained environments like Termux on Android where only older PHP versions may be available.

---

3. Cryptographic Identity

3.1 Key Generation

Each node generates an RSA-2048 key pair at setup time:

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out private.pem
openssl rsa -in private.pem -pubout -out public.pem
```

The private key is stored in `~/.nodepulse/private.pem` (tunnel nodes) or `nodepulse/identity/private.pem` (domain seeds) with permission 0600. It never leaves the node.

3.2 Node ID Derivation

The `node_id` is derived deterministically from the public key:

```
node_id = SHA-256( DER-encoded-public-key )[0:12]
```

This produces a 12-character lowercase hexadecimal string (e.g., `4a7f3c2b1e9d`). The node_id is collision-resistant within the network's expected scale and can be independently verified by any node that holds the corresponding public key.

3.3 Signature Scheme

All announcements are signed using RSA-SHA256. The signing payload is a pipe-delimited string:

Operation	Payload Format
Announce / Publish URL	`node_id|url|signed_at`
Heartbeat	`node_id|heartbeat|signed_at`

The `signed_at` field is an ISO 8601 UTC timestamp (`YYYY-MM-DDTHH:MM:SSZ`). Signatures are base64-encoded.

3.4 Identity Verification

When a node receives an announcement, it performs three verification steps:

1. **node_id derivation check** — Verify that the claimed `node_id` equals `SHA-256(DER(public_key))[0:12]`.
2. **Signature verification** — Verify the RSA-SHA256 signature against the payload and public key.
3. **Timestamp freshness** — For `publish_url` operations, verify that `signed_at` is more recent than the existing entry.

An announcement that fails any of these checks is silently discarded.

---

4. Network Topology

4.1 Node Types

The network comprises two types of nodes:

Tunnel nodes are ephemeral. They run on personal devices (phones, laptops) and expose their local server through a relay tunnel. Their public URL changes on every tunnel restart. They store their identity in `~/.nodepulse/`.

Domain seeds are stable. They run on servers with real domain names and permanent URLs. They store their identity in `nodepulse/identity/` alongside the web-facing code. Domain seeds periodically self-announce to the network to maintain their presence.

4.2 Seed Hierarchy

The network uses three tiers of seeds for bootstrapping and resilience:

Seed Type	File	Description
Origin Seeds	`seeds_origin.json`	Hardcoded bootstrap seeds. Permanently embedded in installation scripts and client code. The safety net of last resort.
Domain Seeds	`seeds_domain.json`	Nodes with real domains that have announced with `type: "domain"`. Discovered dynamically via gossip. Append-only — entries are never pruned.
Network Seeds	`seeds_network.json`	Tunnel nodes with recent liveness (within TTL). Rebuilt dynamically from `directory.json` during each maintenance cycle.

When a new tunnel node starts with an empty peer list, it contacts the origin seeds. From there, gossip propagates the full directory. After the first gossip exchange, the node has enough peers to operate independently of the origin seeds.

4.3 Data Stores

Each node maintains the following JSON files, synchronized through gossip:

File	Contents	Merge Strategy
`directory.json`	Verified registry of all known nodes (with signatures)	Per `node_id`, keep the entry with the most recent `signed_at`
`peers_online.json`	Currently active peers with TTL-based expiry	Per `node_id`, keep the most recent `last_seen`; prune stale entries
`seeds_origin.json`	Hardcoded bootstrap seeds	Static, not merged
`seeds_network.json`	Dynamic tunnel seeds	Rebuilt from directory during maintenance
`seeds_domain.json`	Domain seed registry	Append/update only, never pruned
`node_identity.json`	This node's identity (node_id, public_key, type)	Local only
`node_config.json`	Runtime configuration	Local only

All JSON file I/O uses `flock` — shared locks (`LOCK_SH`) for reads, exclusive locks (`LOCK_EX`) for writes — to prevent corruption from concurrent access.

---

5. Protocol Specification

5.1 API Endpoints

The node API is served at `nodepulse/api.php` with action-based routing via query string:

Endpoint	Method	Description
`?action=announce`	POST	Register a new node or re-register with updated URL. Requires `node_id`, `public_key`, `url`, `type`, `signature`, `signed_at`.
`?action=publish_url`	POST	Update a node's URL. Same fields as announce. Rejects if `signed_at` is not newer than existing entry.
`?action=directory`	GET	Returns full `directory.json`.

Endpoint	Method	Description
`?action=lookup&node_id=xxx`	GET	Lookup a single node by ID. Returns the node's current URL and type.
`?action=peers`	GET	Returns `peers_online.json`.
`?action=heartbeat`	POST	Liveness signal. Requires `node_id`, `signature`, `signed_at`. Updates `last_seen` in directory and peers.
`?action=seeds`	GET	Returns `seeds_origin.json` (for bootstrap).
`?action=update_info`	GET	Returns software version information.

5.2 Announce Flow

When a tunnel node starts:

```
1. cloudflared starts, exposing localhost:8080
2. nodepulse.sh intercepts the tunnel URL from cloudflared's stdout
3. nodepulse.sh signs the announcement: payload = "node_id|url|signed_at"
4. nodepulse.sh POSTs the signed announcement to up to 8 targets
   (selected randomly from known peers + hardcoded seeds)
5. Each target verifies the signature, node_id derivation, and stores the entry
6. The maintenance daemon starts, running gossip cycles every 300 seconds
```

5.3 Rate Limiting

All API endpoints are rate-limited using a file-based system. Rate limit state is stored in `nodepulse/data/ratelimit/` as individual JSON files keyed by `SHA-256(action + ":" + identifier)`.

Default limits (configurable via `node_config.json`):

Action	Limit	Window	Key
announce	5 requests	1 hour	IP address
publish_url	20 requests	1 hour	node_id
heartbeat	120 requests	1 hour	node_id
gossip	30 requests	1 hour	IP address

6. Gossip Protocol

6.1 Overview

NodePulse uses a push-pull gossip protocol for state synchronization. Each node periodically selects a random subset of known peers (the "fanout") and exchanges its full directory and peer list. This achieves eventual consistency across the network without requiring any central coordinator.

6.2 Gossip Endpoint

The gossip endpoint is `nodepulse/gossip.php` (POST only). A gossip request contains:

```
{
  "node_id": "sender's node_id",
  "directory_entries": [ ... ],
  "peers": [ ... ]
}
```

The receiving node:

1. **Validates the sender** — `node_id` must be a valid 12-hex-char string.
2. **Merges directory entries** — For each incoming entry, verifies the RSA-SHA256 signature and `node_id` derivation. If valid, merges by `node_id`: keeps the entry with the more recent `signed_at`.
3. **Merges peers** — For each incoming peer, merges by `node_id`: keeps the more recent `last_seen`. Prunes peers older than the TTL. Caps total peers at `max_peers`.
4. **Filters peers by directory** — Removes any peer whose `node_id` does not appear in the verified directory. This prevents injection of fake node_ids through gossip.
5. **Updates `seeds_domain.json`** — Any directory entry with `type: "domain"` is appended/updated in the domain seed list.
6. **Returns local state** — The response includes the node's own directory entries and peers for bidirectional synchronization.

6.3 Cryptographic Verification in Gossip

Every directory entry accepted through gossip must pass signature verification. This is the critical security property: a node cannot inject a fake entry or claim another node's identity without possessing the corresponding RSA private key.

The merge function `np_merge_directory()` performs two checks on every incoming entry:

```
1. np_verify_signature(payload, signature, public_key)  → RSA-SHA256 check
2. np_verify_node_id(node_id, public_key)               → SHA-256(DER) check
```

Only entries passing both checks are merged. This means the network converges on a consistent, cryptographically verified global directory.

6.4 Convergence

With a default fanout of 3 and a maintenance interval of 100 seconds (lazy cron, triggered by incoming API requests), a new announcement propagates to the entire network in O(log N) gossip rounds. For a network of 1,000 nodes, full propagation typically completes within 3-4 gossip rounds (5-7 minutes).

---

7. Maintenance and Liveness

7.1 Maintenance Cycle

The maintenance engine (`maintenance.php`) performs the following steps:

1. **Peer selection** — Select `gossip_fanout` (default: 3) random peers, excluding self. If no peers are known, fall back to origin seeds.
2. **Gossip exchange** — For each selected peer, POST local directory and peers, merge the response.
3. **Heartbeat broadcast** — Send signed heartbeats to a subset of peers to update `last_seen`.
4. **Rebuild `seeds_network.json`** — From the verified directory, extract tunnel nodes with recent `last_seen` (within TTL), sorted by recency.
5. **Rebuild `seeds_domain.json`** — Append/update domain entries from the directory.
6. **Prune stale peers** — Remove peers from `peers_online.json` whose `last_seen` exceeds the TTL (default: 24 hours).

7.2 Invocation Modes

Maintenance runs in two modes:

Lazy cron — `api.php` and `gossip.php` register maintenance as a `register_shutdown_function`. After serving the HTTP response, PHP checks if the configured interval (default: 100 seconds) has elapsed since the last run. If so, it executes a full maintenance cycle. This eliminates the need for system cron jobs.

CLI — `php maintenance.php` runs a full cycle unconditionally. The standalone daemon `np_maintenance_daemon.php` runs in the background, executing maintenance every 300 seconds and monitoring tunnel health.

7.3 Self-Announce (Domain Seeds)

Domain seeds use `selfannounce.php` to periodically re-announce themselves to the network. The self-announce cycle:

1. Loads the node's identity from `nodepulse/identity/`.
2. Signs the announce payload using PHP's `openssl_sign()`.
3. Collects targets from `seeds_origin.json` and `peers_online.json`, excluding self.
4. Shuffles and limits to 8 targets.
5. POSTs the signed announcement to each target.

The default interval is 1,800 seconds (30 minutes), configurable via `selfannounce_interval_sec` in `node_config.json`.

7.4 Tunnel Health Monitoring

The maintenance daemon (`np_maintenance_daemon.php`) performs health checks on the tunnel URL by requesting `api.php?action=directory`. If 5 consecutive health checks fail (indicating the tunnel is down), the daemon exits with code 1, signaling the parent process to restart the tunnel.

---

8. Recovery System

8.1 Problem Statement

When a tunnel node restarts, the Cloudflare quick tunnel assigns a new random URL (e.g., `https://random-words.trycloudflare.com`). Any visitor with a bookmark or cached link to the old URL will get a connection error. The recovery system ensures these visitors can find the node's new address.

8.2 Architecture

The recovery system has three components:

Service Worker (`sw.js`) — Registered with scope `/`, it pre-caches the beacon pages. For all navigation requests outside `/beacon/`, it uses a network-first strategy: if the network returns a 5xx error or a network error occurs, it serves the cached Recovery Browser instead.

Connectivity Monitor (`nodepulse-sw.js`) — Included in every page via `<script src="/nodepulse-sw.js"></script>`. It pings `/beacon/?ping` every 5 seconds. After 2 consecutive failures, it saves the visitor's current path to `localStorage` and redirects to `/beacon/`.

Recovery Browser (`beacon/`) — A self-contained web application that:

1. Reads the node's `node_id` from PHP-injected bootstrap data or `localStorage`.
2. Builds a seed list from three sources: `localStorage` (most recent data), PHP-injected seeds, and hardcoded fallbacks.
3. Queries each seed via `GET api.php?action=lookup&node_id=xxx`.

4. When a new URL is found, verifies it by pinging `/beacon/?ping` on the new host.
5. If the ping returns `{"ok":true}`, redirects the visitor to the new URL, restoring their original path.

8.3 Progressive Enrichment

While the node is online, the Recovery Browser runs a background enrichment cycle every 5 minutes:

- Fetches `api.php?action=peers` from a random seed to discover new peers.
- Fetches `api.php?action=seeds` to discover new seeds.
- Merges both into `localStorage`, capped at 50 peers.

This broadens the set of targets available for future recovery, reducing dependence on any single seed.

8.4 Cache Strategy

The Service Worker uses two cache buckets:

Cache	Strategy	Contents	Max Age
`nodepulse-static-v6`	Cache-first	Beacon HTML, CSS, JS	365 days
`nodepulse-data-v6`	Network-first	`nodes.json` data	Updated on every online fetch

The 365-day cache lifetime for static assets means the Recovery Browser remains available in the visitor's browser for up to a year after their last visit — even if the tunnel has been down the entire time.

---

9. Security Model

9.1 Threat Model

NodePulse assumes: - Nodes may be malicious and attempt to inject false directory entries. - Network communication is over untrusted channels. - The relay provider (Cloudflare, Tor) is not trusted for identity — only for transport.

9.2 Guarantees

Identity authenticity. A node cannot claim another node's `node_id` without possessing its RSA private key. The `node_id` is cryptographically bound to the public key via SHA-256 derivation.

Entry integrity. Every directory entry includes an RSA-SHA256 signature over `node_id|url|signed_at`. Any modification to the URL or timestamp invalidates the signature.

Replay protection. The `publish_url` action rejects updates where `signed_at` is not newer than the existing entry. This prevents replay of old announcements.

Peer filtering. After gossip merges, peers are filtered against the verified directory. A peer entry without a corresponding verified directory entry (which requires a valid signature) is discarded. This prevents an attacker from injecting arbitrary `node_id`/URL pairs into the peer list.

9.3 Authentication Gate

Individual node applications (terminal, file manager, etc.) are protected by a shared session authentication gate (`auth_gate.php`). The password is stored as a bcrypt hash in `~/.nodepulse/gate_password.hash`. On first access, the user sets a password; subsequent access requires authentication.

9.4 Rate Limiting

File-based rate limiting protects against brute-force attacks on the announce and heartbeat endpoints. Rate limit state is stored as individual JSON files with SHA-256 hashed filenames, preventing enumeration.

9.5 CORS Policy

The API uses `Access-Control-Allow-Origin: *` by design. This is intentional: the Recovery Browser running on an old tunnel URL must be able to query seeds on different domains to find the node's new URL. The security boundary is the cryptographic signature, not the origin.

---

10. Supported Platforms

NodePulse runs on any platform with bash, PHP (5.6+), OpenSSL, and curl:

Platform	Environment	Setup Script	Notes
Android	Termux	`install_termux/termux-setup.sh`	PHP + lighttpd via `pkg`.

Platform	Environment	Setup Script	Notes
Windows	MSYS2 (UCRT64)	`install_msys2/msys2-setup.sh`	standalone PHP, nginx, cloudflared binaries.
Linux	Native	Manual	Any distro with PHP.
macOS	Native	Manual	PHP via brew.

10.1 MSYS2 Setup

The MSYS2 setup script (`msys2-setup.sh`) performs a fully automated installation:

1. Installs base tools via pacman (curl, unzip, openssl).
2. Installs Python + packages for optional features.
3. Downloads PHP for Windows from `windows.php.net`.
4. Downloads nginx from `nginx.org`.
5. Downloads `cloudflared` from GitHub.
6. Configures `php.ini` (curl, openssl, mbstring extensions).
7. Configures nginx (port 8080, FastCGI to php-cgi on port 9000).
8. Generates RSA-2048 identity and derives `node_id`.
9. Installs management scripts: `start-server`, `stop-server`, `server-status`.

All binaries are installed to `~/nodepulse-bin/`. The web root is `~/www/`.

10.2 Termux Setup

The Termux setup follows the same flow but uses `pkg` for package management and the PHP built-in server or lighttpd instead of nginx. The identity is stored in `~/.nodepulse/` with the same format as MSYS2.

---

11. Relay Abstraction

11.1 Principle

The relay mechanism is entirely outside the protocol. NodePulse signs and announces a URL — it does not care how that URL was obtained. The `nodepulse.sh` script intercepts the relay's output to capture the public URL, then feeds it into the standard announce flow.

11.2 Cloudflare Tunnel (Current Default)

```
cloudflared tunnel --url http://127.0.0.1:8080
```

The script parses stdout for `https://xxx.trycloudflare.com` and uses it as the announce URL. Free quick tunnels assign a new URL on every restart. The announce-gossip-recovery cycle handles this transparently.

11.3 Planned Relay Providers

Provider	URL Stability	Anonymity	Speed	Cost
Cloudflare Tunnel	Ephemeral (changes on restart)	No	High	Free
Tor Hidden Service	Permanent (.onion derived from key)	Full	Medium	Free
frp on VPS	Permanent (subdomain from node_id)	No	High	VPS cost
rathole on VPS	Permanent	No	High	VPS cost

Tor integration eliminates the URL change problem entirely: the `.onion` address is derived from a cryptographic key and remains stable indefinitely. Only heartbeat announcements are needed for liveness.

VPS-based relays (frp/rathole) use the first 6 hex characters of `node_id` as a subdomain (e.g., `4a7f3c.plexum.net`), providing a human-readable, stable URL tied to the node's cryptographic identity.

---

12. Software Update Mechanism

NodePulse includes a software update system via `update/update.json`:

```
{
  "current_version": "1.0.0",
  "release_date": "2026-03-02T00:00:00Z",
  "download_url": "downloads/apps.zip",
  "sha256": "",
  "signature": "",
  "changelog": "Initial release"
}
```

Nodes with `auto_update: true` in their configuration can query any seed's `?action=update_info` endpoint to check for new versions. The update package is verified by SHA-256 hash and (in future versions) RSA signature before installation.

---

13. Future Work

13.1 Multi-Relay Support (Phase 5)

An optional `relay_provider` field will be added to peer entries to enable relay-aware routing. Nodes with Tor support will automatically route requests to `.onion` peers through SOCKS5 proxy. The field is informational and does not affect the signing payload, maintaining full backward compatibility.

13.2 Decentralized Marketplace (Phase 6)

A listing system propagated via gossip, enabling peer-to-peer commerce without central servers:

- **Listings** — Signed by the seller, propagated via gossip, with automatic TTL-based expiry.
- **Categories** — Extensible taxonomy (services/design, goods/digital, etc.).
- **Merge strategy** — Per `listing_id` + `version`, highest version wins.

13.3 Escrow System (Phase 7)

A custodial 2-of-3 voting system using existing RSA keys:

- Three parties: buyer, seller, arbiter (selected algorithmically from a gossip-propagated arbiter registry).
- Funds are released when 2 of 3 parties cast concordant votes.
- The arbiter alone cannot release funds — a vote from one of the transacting parties is always required.
- Timeout mechanism: automatic refund to buyer after 14 days without resolution.
- MVP uses signed IOUs; future versions integrate Lightning Network for real payments.

13.4 Reputation System

Transaction receipts with mutual ratings, signed by both parties, propagated via gossip:

- Each completed transaction generates a receipt with buyer and seller ratings (1-5).
- Receipts require both signatures to be valid, preventing self-attribution.

- Reputation is computed locally by each node from the gossip-propagated receipt set.

13.5 Lightning Network Integration (Phase 8)

Replacement of IOUs with real Bitcoin Lightning payments, using LND or LNbits as a local wallet. The escrow system generates Lightning invoices for payment locking and release.

---

14. Conclusion

NodePulse demonstrates that a decentralized, self-healing node registry can be built with minimal infrastructure requirements. By combining RSA-SHA256 cryptographic identity, gossip-based state propagation, and browser-side recovery via Service Workers, the protocol achieves:

- **Permissionless participation** — Any device with PHP and an internet connection can join.
- **Cryptographic identity persistence** — Node identity survives across URL changes, device restarts, and relay migrations.
- **Self-healing discovery** — Visitors never permanently lose a node; the Recovery Browser re-discovers it automatically.
- **Relay independence** — The protocol is agnostic to the transport layer; nodes can freely switch between Cloudflare, Tor, or VPS-based relays.
- **Zero central authority** — After bootstrap, the network operates through peer-to-peer gossip without any central server dependency.

The planned marketplace, escrow, and reputation systems extend these properties into peer-to-peer commerce, using the same cryptographic primitives and gossip infrastructure already in place.

Network: Plexum — Software: NodePulse Version: 1.0.0