Netfory

A Peer-to-Peer Electronic Application and Data Distribution Network (Web 4.0 Protocol)

Alternative Academic Title:
Netfory: A Decentralized, Agent-Native Infrastructure for Zero-Infrastructure Web 4.0

Authors:
TechnoL0g & Europa · SmartHoldem Core Team

Date: July 2026
Version: 1.0.0
Network: SmartHoldem (STH) — DPoS Blockchain, secp256k1
Token: STH (1 STH = 10^8 smarthoshi)

Core Stack:
Tauri v2 (Rust) · Vue 3 (TypeScript) · Iroh (QUIC) · BLAKE3 · sled KV

Contact:

- Web: https://smartholdem.io https://netfory.com
- X / Twitter: @smartholdem @thenetfory
- Telegram: @smartholdem @netfory

Taglines:

- *"The Web That Works For You."*
- *"Don't visit the web. Be the web."*
- *"Zero Infrastructure. Absolute Autonomy."*

Abstract (Summary)

Modern decentralized applications suffer from a fatal vulnerability: the centralization of frontend hosting. While smart contracts reside on blockchains, the user-facing interfaces remain dependent on AWS, Cloudflare, and Vercel. A single regulatory action can erase any "decentralized" application from the internet. Existing peer-to-peer alternatives lack either the performance, the economic incentives, or the human-readable addressing required for mass adoption.

Netfory is a fully autonomous, agent-native Web 4.0 infrastructure. It replaces centralized hosting with a decentralized, content-addressed distribution network powered by the Iroh (QUIC) transport layer and BLAKE3 cryptographic verification. To guarantee absolute censorship resistance and eliminate single points of failure, the protocol enforces two core architectural invariants: **trackerless peer discovery via Mainline DHT** and **resilient, high-speed content distribution via a Multi-Armed Bandit swarm algorithm**. Through a native Proof-of-Utility economic model, network participants (seeders) are directly compensated in STH for providing storage and bandwidth. Developers and users interact through censorship-resistant identities (`u://` and `.sth`) without intermediaries.

The network is not managed by agents. The network **IS** the agents.

This is not a concept. It is a working, tested, production-ready stack.

---

1. Abstract

Commerce on the internet has come to rely almost exclusively on financial institutions and centralized hosting providers serving as trusted third parties to process electronic payments and serve application frontends. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust-based model.

Modern Web 3.0 applications face a fatal architectural contradiction. Smart contracts and tokens reside on decentralized blockchains, but the user interfaces through which humans interact with them are hosted on centralized infrastructure — AWS, Cloudflare, Vercel, Google Cloud. A single court order, a single provider's terms-of-service violation, or a single government firewall can erase any "decentralized" application from the global internet. The frontend is the single point of failure.

Existing peer-to-peer alternatives do not fully resolve this problem. They lack either the performance, the economic incentives, or the human-readable addressing required for mass adoption. None of these systems close the economic loop required to sustain a permanent, censorship-resistant application layer.

We propose a solution to the decentralized hosting crisis: **Netfory**, a fully autonomous, agent-native Web 4.0 infrastructure built on top of the SmartHoldem (STH) DPoS blockchain.

Netfory replaces the centralized hosting stack with a peer-to-peer application and data distribution network. Applications (dApps) are ingested, hashed with BLAKE3, signed by their authors with secp256k1 keys, and distributed as immutable content-addressed blobs via the Iroh (QUIC) transport layer. Users access applications through human-readable, censorship-resistant identifiers: `u://username` for identities, `<name>.sth` for decentralized domains, and `sth://<appId>` for applications. Resolution is anchored on-chain via a first-win registration model.

The native token STH is integrated directly into the protocol as fuel and anti-spam shield. Registration of a name costs 1 STH. Publication of a dApp costs 100 STH (dynamic by payload size). A message to a stranger costs `msg_cost_sth`, set by the receiver. These fees make large-scale spam and Sybil attacks economically devastating:

$$C_{spam} = N \times msg\_cost\_sth$$

For $N = 1{,}000{,}000$ messages at 0.01 STH each, the attacker burns $10{,}000$ STH — a provably irrational expenditure.

In return, network participants who provide storage and bandwidth are compensated through a native Proof-of-Utility model. Seeders broadcast signed heartbeats every 5 minutes. Trackers maintain a 24-hour sliding window (288 time slots) and compute an uptime coefficient K_{up}. Seeder income is calculated as:

$$Estimated = (Base + seededGb \times density + activeApps \times PerApp) \times K_{up}$$

Money flows directly from users to creators of value — authors, developers, seeders. There is no platform tax. There is no centralized intermediary. The economy is a perpetual, self-sustaining loop.

Netfory is not a theoretical roadmap. It is a live, production-ready stack anchoring identity, distribution, and economics on the SmartHoldem DPoS blockchain. The remainder of this paper details the architecture that makes this possible.

---

2. Introduction

2.1 The Frontend Paradox of Web 3.0

The current generation of decentralized applications suffers from a fatal architectural contradiction. Smart contracts, tokens, and on-chain logic reside on distributed ledgers — immutable, censorship-resistant, and globally available. Yet the user interfaces through which humans interact with these contracts are hosted on centralized infrastructure: Amazon Web Services, Cloudflare, Vercel, Google Cloud.

This is the **frontend paradox**. A single court order, a single provider's terms-of-service violation, or a single government firewall can erase any "decentralized" application from the global internet. The smart contract survives, but the door to it is locked. The frontend is the single point of failure.

Consider the practical consequence: a decentralized exchange hosted on Vercel can be deplatformed in under 60 seconds. A governance dashboard on AWS can be seized by regulatory action. A wallet interface served through Cloudflare can be blocked by a national firewall. The blockchain remains, but users cannot reach it. Decentralization ends at the browser.

This vulnerability is not theoretical. It has been demonstrated repeatedly across jurisdictions. The promise of Web 3.0 — user ownership of data and assets — is undermined by centralized control of the access layer.

2.2 Defining Ideal Decentralization

We define **ideal decentralization** as a system or network in which no single entity has control or the ability to make decisions for the entire system. Power and decision-making are distributed among multiple participants. No registrar can revoke an identity. No host can remove an application. No intermediary can intercept a message.

Ideal decentralization requires four properties:

1. **Content-addressability.** Data is identified by cryptographic hash, not by location. A file exists because it is verified, not because a server says so.
2. **Censorship-resistance.** No single actor can prevent publication, distribution, or access.
3. **Economic self-sufficiency.** Participants who provide value (storage, bandwidth, computation) are directly compensated without intermediary extraction.
4. **Autonomy.** The system operates without dependence on external infrastructure. If the global internet fragments, the local network persists.

Web 3.0 achieves partial decentralization. It decentralizes the ledger but not the access layer. It decentralizes ownership but not distribution. It decentralizes finance but not hosting.

Netfory achieves ideal decentralization across all four properties. The application, the identity, the economy, and the transport are all native to the peer-to-peer layer.

2.3 The Evolution: Web 2.0 → Web 3.0 → Web 4.0

The internet has evolved through three distinct paradigms:

Web 2.0 — The Platform Era. Users read and write. Data resides on corporate servers. Identity is granted by platforms. Value flows to intermediaries. Examples: social networks, cloud storage, centralized marketplaces.

Web 3.0 — The Ownership Era. Users read, write, and own. Tokens and smart contracts reside on blockchains. Yet the frontend remains centralized. Identity is often tied to OAuth providers or ENS subscriptions. Value flows to protocols but hosting fees flow to AWS.

Web 4.0 — The Symbiotic Era. Users read, write, own, and *are*. The network is not a place you visit; it is an infrastructure you inhabit. Every client is an autonomous agent. Every device is a node. Every action is a protocol interaction, not an API call.

Web 4.0 is defined by four foundational formulas:

1. **Protocol Stack replaces API.** Access is a matter of connecting to a protocol (`sn://`, `u://`, `sth://`, `dev://`), not having permission on a server.
2. **State-First Distribution.** State exists distributed in the network. The user does not "download" a page; they synchronize state via P2P.
3. **Zero-Infra.** No AWS, no Google Cloud, no Vercel. All backend logic — databases, APIs, load balancers — migrates to the P2P protocol layer.
4. **Content-Addressability Everywhere.** Everything is bound to a CID. Content proves itself. No domain can be seized. No certificate can be revoked.

A critical correction to the prevailing narrative must be stated explicitly: **the network is not managed by agents. The network IS the agents.** There are no server-based "manager agents" orchestrating routing or resource allocation. Every user's client is itself an autonomous node that discovers peers (mDNS locally, n0-discovery globally), self-verifies content integrity (BLAKE3), negotiates routes directly with other peer-agents (Iroh QUIC + Gossipsub), and self-sustains the economy via signed heartbeats.

The user's device is the agent.

2.4 Why Existing P2P Architectures Fall Short

Existing peer-to-peer systems do not fully resolve the crisis of centralized hosting. Each suffers from structural limitations: performance bottlenecks, absence of economic incentives, lack of human-readable addressing, or dependency on external infrastructure. A detailed comparative analysis is provided in §3.4.

Netfory unifies all four requirements: content-addressed distribution (Iroh + BLAKE3), economic incentives (STH token + Proof-of-Utility), human-readable identity (`u://` and `.sth` on SmartHoldem DPoS), and a native execution environment (Tauri v2 + Vue 3 with `window.smartholdem` bridge).

2.5 Real-World Scenarios

To illustrate the practical impact of ideal decentralization, consider the following scenarios:

Alice publishes a poker dApp. She packages her Vue 3 application, signs it with her secp256k1 key, and publishes it to the Netfory store for 100 STH. The application is hashed with BLAKE3, assigned a content identifier, and distributed as an immutable blob via Iroh. She registers `poker.sth` on-chain, binding the human-readable name to her `app_id`. The application is now available at `sth://<appId>` or `poker.sth`. It cannot be removed. It cannot be censored. It cannot be deplatformed. As long as one seeder in the world holds the blob, the application exists.

Bob distributes a music album. He uploads 12 tracks as a single content-addressed collection. Each track is verified by BLAKE3. He announces the album via `sn://album/<id>`. Fans download directly from peers. Bob earns STH from paid downloads. Seeders earn STH from distributing the tracks. No platform takes a cut. No intermediary controls distribution.

Charlie hosts source code on DevHub. He initializes a repository with `git init` and pushes to `dev://charlie/smart-swarm`. The `git-remote-dev` helper packages commits into encrypted Iroh collections, signs them with his private key, and updates the branch pointer in Mainline DHT. His IDE (VS Code, WebStorm, GitKraken) works without modification. The repository lives as long as one peer holds it. GitHub cannot ban him. A government cannot seize the repo. The code is immortal.

Diana streams a movie. She opens `sn://file/<cid>` in her client. The movie is streamed directly from peers via Iroh QUIC. No central server. No buffering delays from a single source. The content is verified chunk-by-chunk by BLAKE3. If one seeder goes offline, the stream seamlessly switches to another.

Eve chats privately with Frank. She opens a P2P chat via `u://frank`. Messages are encrypted end-to-end with X25519 (ECDH) key exchange and AES-256-GCM. Messages travel directly via Iroh Gossipsub. No central server stores the messages. No metadata is collected. If Eve sets `msg_cost_sth = 0.1`, strangers must pay 0.1 STH to message her — spam becomes economically devastating.

These scenarios are not hypothetical. They are functional today in the Netfory stack.

2.6 The Netfory Proposition

Netfory is a fully autonomous, agent-native Web 4.0 infrastructure built on the SmartHoldem (STH) DPoS blockchain. It replaces centralized hosting with a decentralized, content-addressed distribution network powered by the Iroh (QUIC) transport layer and BLAKE3 cryptographic verification.

Through a native Proof-of-Utility economic model, network participants (seeders) are directly compensated in STH for providing storage and bandwidth. Developers and users interact through censorship-resistant identities (`u://` and `.sth`) without intermediaries.

The network is not managed by agents. The network **IS** the agents.

This is not a concept. It is not a roadmap. It is a working, tested, production-ready stack: a Tauri v2 desktop client, a headless Rust seeder, a lightweight sled-based tracker, a `git-remote-dev` helper for decentralized source code, and a DPoS blockchain anchoring identity and economics.

The remainder of this paper details the architecture, the transport layer, the identity system, the developer ecosystem, the economic model, and the security properties that make Netfory the first implementation of Web 4.0.

---

3. The Network Core & Transport Layer

3.1 The Three Components

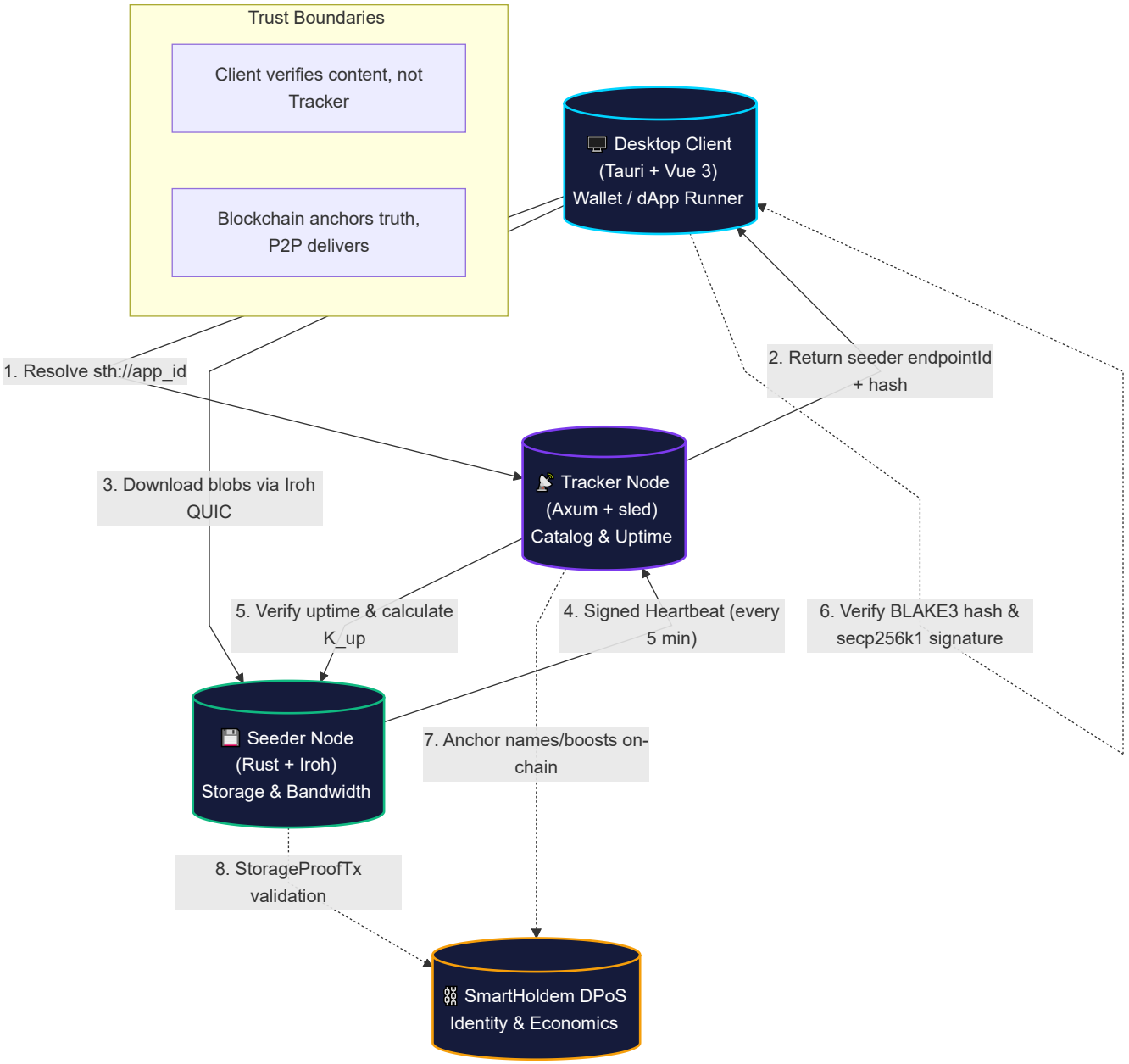
The Netfory network is composed of three distinct but interoperable components. Each is implemented as a standalone Rust binary or Tauri application, communicating over a unified wire protocol.

1. Tracker. A lightweight HTTP coordinator built on `axum 0.8` and the `sled` embedded key-value database. The tracker does **not** store content. It maintains an in-memory registry of signed announcements with TTL-based heartbeats, mapping `app_id` → `content_hash` → `seeders`. It computes per-node uptime via signed heartbeats and exposes a minimal HTTP API (`/apps`, `/seeder/heartbeat`, `/seeder/heartbeat/<address>`). The tracker is a discovery aid, not a trust anchor. Content integrity is verified by the client via BLAKE3, not by the tracker.

2. Seeder. An autonomous headless node (Rust, Tokio async runtime) deployed on VPS or dedicated hardware. The seeder connects to the `iroh-blobs` swarm, fetches content-addressed tar archives of dApps from providers, and redistributes identical blobs 24/7. It exposes a Prometheus-compatible `/metrics` endpoint for the Network Map leaderboard and broadcasts signed heartbeats to trackers every 5 minutes. The seeder is the physical backbone of the network.

3. Desktop Client. A cross-platform GUI application built on Tauri v2 (Rust) + Vue 3 (TypeScript). The client is simultaneously a cryptocurrency wallet, a decentralized store gateway, a dApp execution environment, and a hybrid seeder. When the user enables "Earn STH", the client begins redistributing installed dApps and broadcasting signed heartbeats, functioning as a full peer in the network.

All three components speak the same wire protocol. The desktop client and the headless seeder are functionally equivalent peers; the only difference is the presence of a GUI.



3.2 Transport Layer: Iroh (QUIC)

Netfory replaces the legacy libp2p/IPFS stack with **Iroh**, a modern peer-to-peer transport layer built on the QUIC protocol. This architectural choice delivers four critical advantages:

1. Direct P2P through strict NATs. Iroh implements STUN/TURN/DERP hole-punching, enabling direct peer-to-peer connections even behind carrier-grade NATs and hardware firewalls. No port forwarding is required. No public IP is exposed. Seeders can operate from residential connections without DDoS risk.

2. Zero-RTT connection resumption. QUIC eliminates the latency penalty of TCP's 3-way handshake plus TLS negotiation. Connection establishment requires a single round trip:

$$L_{TCP} = RTT \times 3 \quad (\text{SYN} + \text{SYN-ACK} + \text{ACK} + \text{TLS})$$

$$L_{QUIC} = RTT \times 1 \quad (\text{0-RTT resumption})$$

For a transatlantic connection with $RTT = 80$ ms, TCP requires 240 ms per new connection. QUIC requires 80 ms — a **3× reduction** in connection latency. For a dApp that opens 50 parallel streams to fetch chunks, this translates to seconds saved per load.

3. Async multiplexed streams. QUIC natively supports multiple independent streams over a single connection, with no head-of-line blocking. If one stream is delayed by packet loss, other streams continue uninterrupted. This is critical for real-time dApp loading, where a single stalled chunk must not block the rendering of the entire application.

4. mDNS for LAN-first autonomy. Iroh integrates multicast DNS (mDNS) for instant peer discovery within local networks. If a city or region is disconnected from the global internet (via TSPU, cable cut, or natural disaster), Netfory nodes automatically discover each other via mDNS and continue operating. Clearnet, Tor, and I2P die in this scenario. Netfory survives.

The transport layer also includes a **decentralized DERP relay mesh** — implemented as `n1-relay`, a lightweight Rust service for relaying encrypted traffic between peers. The relay helps bypass strict NATs and hides real IP addresses. The relay is decentralized — any peer can operate a relay node — and is used only when direct hole-punching fails (e.g., symmetric NAT). Crucially, the relay never sees plaintext traffic; it forwards only encrypted QUIC packets, preserving end-to-end confidentiality.

3.3 Content Addressing with BLAKE3

All content in Netfory is addressed by its **BLAKE3 cryptographic hash**, not by its location. A dApp, a file, a library, a git commit — each is identified by a content identifier (CID) derived from BLAKE3. This is the foundation of censorship resistance: no domain can be seized, no certificate can be revoked, no server can be deplatformed. Content proves itself.

BLAKE3 was selected over SHA-256 for a decisive performance reason:

$$V_{BLAKE3} \geq 10.0 \text{ GB/s}$$

$$V_{SHA-256} \approx 0.4 \text{ GB/s}$$

$$\frac{V_{BLAKE3}}{V_{SHA-256}} \geq 25\times$$

BLAKE3 is **25× faster** than SHA-256. For a seeder verifying 100 GB of content per hour, BLAKE3 completes the verification in under 10 seconds. SHA-256 would require over 4 minutes — a 24× increase in CPU overhead. At network scale, this difference determines whether real-time integrity verification is feasible or a bottleneck.

Content is stored as fragmented, encrypted blobs sliced into chunks and verified by BLAKE3 hash trees. The seeder operator physically cannot see what data they store — the blobs are opaque to the host. If the community marks an `app_id` as malicious via DPoS voting, trackers stop announcing the hash, and seeders auto-purge the blobs via eviction policy.

3.4 Comparative Analysis

The following table compares Netfory against existing architectures across the critical dimensions of a decentralized application network.

Criterion	Clearnet (Web 2.0/3.0)	Tor / I2P	IPFS	Freenet (Hyphanet)	Netfory (Web 4.0)
Transport	HTTP/HTTPS over TCP/IP. Fully centralized routing.	Onion (TCP) or garlic (UDP/SSU) routing through relay nodes.	Bitswap, Kad-DHT over libp2p. Content-addressed.	Specialized P2P routing by key.	**Iroh (QUIC + Gossipsub + Blobs).** Direct async P2P with native NAT-traversal.
Frontend Hosting	Centralized servers (AWS, Cloudflare, Vercel). Censorable.	Hidden services / eepsites. Require constant uptime of creator's server.	Centralized IPFS gateways. Dependent on Web2 infra.	Static Freesites in distributed DB. High latency.	**Native autonomous client (Tauri v2 + Vue 3).** dApps distributed as immutable iroh-blobs directly between peers.
Identity	Logins, passwords, OAuth, phone numbers. Tied to person.	Cryptographic hashes of keys. Long unreadable strings.	Cryptographic PeerID. No human-readable profiles.	Cryptographic keys (SSK, USK). Complex for users.	**Decentralized on-chain identity `u://name`.** First-win registration on SmartHoldem DPoS. One seed controls all.
Economy & Anti-Spam	None. Moderation via CAPTCHAs, IP bans.	None. Vulnerable to DoS and Sybil attacks.	None at base layer. Free pinning unreliable.	None. Data evicted by LRU if unpopular.	**Native STH token integrated at protocol level.** Paid names, paid dApps, paid messages. Spam is economically devastating.
Domain Resolution	Hierarchical ICANN/DNS. Censorable, seizable.	Internal hash domains (.onion, .i2p). Unmemorable.	IPNS/ENS (requires Ethereum, expensive gas) or centralized DNS-TXT.	Internal keyspace (USK). Strictly intra-network.	**Decentralized dDNS (.sth domains).** Full user control, instant resolution, built-in P2P marketplace.
Messaging	Centralized platforms (Telegram, WhatsApp). Metadata on servers.	Third-party utilities (I2P-Bote, Ricochet). Slow, no presence.	None out of the box. Requires overlays.	FMS (Freenet Messaging System). Slow forums, hours of delay.	**Built-in E2EE P2P messenger (X25519 + AES-256-GCM).** Instant transit via Gossipsub, delivery indicators (✓/✓✓), on-chain anti-spam.
LAN Autonomy	Fully non-functional without external internet or DNS.	Cannot function without global directories and introducers.	Possible with direct connection, but resolution difficult.	Does not work in isolated networks. Depends on global network volume.	**Full autonomy.** Automatic peer discovery via mDNS. Chats, dApps, and file sharing work in LAN without internet.

Netfory is the only architecture that simultaneously provides: content-addressed distribution, economic incentives for long-term availability, human-readable identity anchored on a blockchain, a native execution environment for dApps, end-to-end encrypted messaging, and LAN-first autonomy.

3.5 Technical Execution Flows

To demonstrate the mechanical superiority of the Netfory transport layer, consider three specific technical execution flows that are impossible or highly inefficient in legacy P2P systems.

Flow 1: Headless Seeder behind Strict Carrier-Grade NAT
A seeder operator deploys a headless node on a residential connection behind a strict symmetric NAT, with no public IP and no port forwarding capabilities.

1. The seeder initiates an Iroh endpoint using the N0 preset.
2. It contacts a decentralized DERP (Designated Encrypted Relay for Packets) mesh to perform STUN-based hole-punching.
3. A desktop client requests a dApp blob. The tracker provides the seeder's `endpointId`.
4. The client and seeder coordinate via the DERP relay to establish a direct, encrypted QUIC connection, bypassing the NAT without exposing the seeder's real IP address to the public internet.
5. Result: Zero-infrastructure hosting with built-in DDoS protection.

Flow 2: Real-Time dApp Streaming with Zero Head-of-Line Blocking
A user launches a 500 MB dApp. The client requests the blob via Iroh.

1. The dApp is split into 64 KB chunks, each independently addressed by a BLAKE3 hash.
2. The client opens multiple concurrent QUIC streams to fetch chunks from 5 different seeders simultaneously.
3. If a packet is dropped on Stream 3 due to transient network congestion, QUIC's independent stream multiplexing ensures that Streams 1, 2, 4, and 5 continue delivering data uninterrupted.
4. As each 64 KB chunk arrives, the client computes its BLAKE3 hash in <1 millisecond. If it matches the manifest, it is immediately passed to the Tauri WebView for execution.
5. Result: The dApp begins rendering before the full 500 MB is downloaded, with no TCP-style head-of-line blocking.

Flow 3: Regional Internet Blackout (LAN Autonomy)
A regional ISP suffers a catastrophic fiber cut, or a state actor activates deep packet inspection (TSPU) to block external P2P traffic.

1. Global tracker connectivity is lost. n0-discovery fails.
2. Netfory clients on the local network automatically fall back to multicast DNS (mDNS).
3. Within seconds, devices broadcast their `endpointId` and available `app_id` hashes to the local subnet (e.g., `192.168.1.x`).
4. Users discover each other, establish direct QUIC connections over the LAN, and continue to chat via E2EE, share files via CID, and run locally cached dApps.
5. Result: The network degrades gracefully to a fully functional, autonomous local mesh, while Clearnet, Tor, and IPFS gateways cease to operate.

---

4. Identity, Naming & Social Layer

4.1 Self-Decentralized Identity (SDI)

In traditional systems, identity is granted by centralized authorities: OAuth providers, email registrars, phone number verifiers. These entities can revoke access, collect metadata, and censor users at will. Netfory eliminates this dependency through Self-Decentralized Identity (SDI).

SDI is a cryptographic identity system in which the user is the sole owner of their keys, names, and data. There is no server-side account to ban. There is no central database to breach. Identity is defined by possession of a private key, secured locally, and anchored on the SmartHoldem DPoS blockchain.

The identity system is built on three cryptographic primitives:

1. **Master Seed.** A 12-word BIP-39 mnemonic phrase generates the root entropy. This seed is the single source of truth for all keys and addresses. Loss of the seed means permanent loss of access — there is no "forgot password" recovery.
2. **Hierarchical Derivation.** Using BIP-32 and BIP-44 standards, the client derives independent key pairs for each subsystem (wallet, messaging, dApp signing). For example, a poker dApp receives its own isolated address derived via path `m/44'/255'/0'/1/i`, protecting user privacy while maintaining unified key management.
3. **Local Encryption.** The entire key vault is encrypted on the user's device using AES-256-GCM with a user-defined PIN or password. The seed never exists in plaintext on disk. All cryptographic operations execute in a background thread (`spawn_blocking`) to prevent UI freezing.

4.2 Address Format & Cryptography

Netfory uses the secp256k1 elliptic curve for all cryptographic operations, ensuring compatibility with the SmartHoldem blockchain and Bitcoin-derived standards.

Address Format:

- Curve: secp256k1
- Address prefix: `S` (base58check encoding)
- Derivation: `version_byte || RIPEMD160(SHA256(compressed_pubkey))`
- Example: `S...` (34 characters)

Key Properties:

- **Unforgeable.** Deriving a private key from a public address requires solving the elliptic curve discrete logarithm problem — computationally infeasible with current technology.
- **Deterministic.** The same seed always produces the same addresses. Users can restore their full identity on any device by importing the 12-word mnemonic.
- **Private by default.** Public addresses are pseudonymous. No personal information (name, email, phone) is required or collected.

4.3 Procedural Avatars

Visual identity in Netfory is generated mathematically, not uploaded. Each user's avatar is a **procedural identicon** derived from the hash of their public key.

Generation Algorithm:

1. Compute BLAKE3 hash of the compressed public key.
2. Use the first 32 bytes of the hash as a seed for a deterministic pseudo-random number generator.
3. Render a symmetric geometric pattern (e.g., 5×5 grid with rotational symmetry) using the generated values.

Properties:

- **Unforgeable.** Forging an avatar requires producing a public key with the same hash — equivalent to breaking BLAKE3.
- **Unique.** No two users can have the same avatar unless they control the same private key.
- **Zero bandwidth.** Avatars are generated client-side. No images are transmitted over the network.

This eliminates the need for profile picture uploads, CDN storage, and the associated privacy risks. The avatar is cryptographically bound to the identity.

4.4 Naming System: u:// and .sth

Human-readable addressing is essential for mass adoption. Users cannot memorize 34-character addresses or 64-character hashes. Netfory provides two layers of decentralized naming:

1. u://username — On-Chain Identity

- Registered via a single on-chain transaction on SmartHoldem DPoS.
- The name is stored in the `vendorField` of the transaction.
- An indexer scans the blockchain and records `username → address` mappings.
- **First-win rule.** The first user to register a name owns it permanently. No transfers, no auctions, no expiration.
- **Reserved names.** System-critical names (`nameservice`, `smartnet`, `admin`, `root`, `system`, `support`, `group`, `groups`) are hardcoded and cannot be registered. The validation occurs at the moment of transaction commit — even a direct transaction from an external wallet will be rejected by the indexer.

2. .sth — Decentralized Domains

- Bound to an `app_id` and developer address via on-chain transaction.
- Functions as a human-readable alias for `sth://<appId>`.
- Tracked by the `smartnet-tracker`, which monitors blockchain blocks and caches `name → app_id → content_hash` mappings in the local `sled` database.
- Resolution is instant: when a user types `poker.sth` in the address bar, the client queries the tracker, retrieves the content hash, and begins streaming the dApp via Iroh.

Cost:

- Registration of `u://username`: 1 STH (anti-squatting fee).

- Registration of `<name>.sth`: dynamic fee based on name length and demand.

Both names are **censorship-resistant**. No registrar can revoke, block, or redirect them. The blockchain transaction is the highest mathematical proof of ownership.

4.5 URI Schemes & Namespace Separation

Netfory enforces strict separation of address spaces to prevent namespace collisions and phishing attacks.

Scheme	Purpose	Example
`u://<username>`	User identity (on-chain name)	`u://alice`
`<name>.sth`	Decentralized domain (bound to app_id)	`poker.sth`
`sth://<appId>`	Installed dApp (content-addressed)	`sth://<id>/index.html`
`dev://<username>/<project>`	Git repository (DevHub)	`dev://technolog/smart-swarm`
`sn://<service>`	Native system pages	`sn://nameservice`, `sn://dashboard`
`file/<cid>`	File by content identifier	`file/<cid>~~<node>~~<size>~~<price>~~<owner>~~<name>`

Namespace Rules:

- `u://<username>` resolves to a public profile page.
- `sn://` aliases are reserved for system pages (dashboard, settings, network map) and cannot be hijacked by user-registered names.
- `sth://` is reserved for content-addressed dApps.
- `dev://` is reserved for decentralized source code repositories.

This separation ensures that a user cannot register `u://dashboard` and impersonate the system page. The routing logic enforces strict precedence: system aliases (`sn://`) take priority over user names (`u://`).

4.6 Profile & Social Graph

Each user has a **public profile** — a decentralized social page visible to the entire network.

Profile Contents:

- Username (`u://name`)
- Procedural avatar (identicon)
- Public key (for E2EE messaging)
- Balance and contribution statistics (dApps published, bytes seeded)
- Social graph: followers and subscriptions

Subscriptions:

- Subscribing to a user (`sub:<username>`) is a paid on-chain operation.
- The subscriber sends STH directly to the author — no platform intermediary.
- Subscriptions serve dual purposes: anti-spam (costly to mass-follow) and creator monetization (recurring revenue).
- Verification badge: users with 10+ subscribers receive a verification indicator.

Social Graph Properties:

- **Decentralized.** The follower/subscriber list is stored locally and synchronized via P2P gossip.
- **Private by default.** Users can set their profile to private, hiding the social graph from public view.
- **Censorship-resistant.** No platform can remove a user's profile or delete their followers.

4.7 E2EE Messaging & Key Exchange

Netfory includes a built-in end-to-end encrypted (E2EE) messenger operating directly over the P2P transport layer.

Key Exchange:

- Algorithm: X25519 (Elliptic Curve Diffie-Hellman)
- Each user publishes their X25519 public key and Iroh `endpointId` on-chain via transaction: `xkey:<pub>|<nodeid>`
- This enables any user to initiate a secure conversation by looking up the recipient's public key from the blockchain.

Packet Encryption:

- Algorithm: AES-256-GCM
- For private chats: the room key is derived from the shared ECDH secret.
- For group chats: the room key is derived from a shared group secret (distributed via invite link).

Message Transit:

- Messages travel directly via Iroh Gossipsub — no central server stores or relays them.
- Bootstrap: the sender looks up the recipient's `endpointId` from the blockchain and establishes a direct QUIC connection.
- Delivery indicators: ✓ (sent) and ✓✓ (delivered when peer is online).

Anti-Spam:

- If a user sets `msg_cost_sth > 0`, strangers must pay the specified amount in STH to send a message.
- The payment is an on-chain transaction with encrypted payload: `msg:<ciphertext>`.
- This makes spam economically devastating and monetizes the recipient's attention.

Group Chats:

- Created via invite link: `sn://group/<secret>~~<node>~~<name>~~<owner>`
- E2EE on shared secret: key = `SHA-256("v1" || secret)`, topic = `SHA-256(secret)`
- Self-healing mesh: bootstrap nodes are extracted from the link and remembered via `NeighborUp`

events — after restart, participants rediscover each other.
- Admin panel: only the group owner (address embedded in the link) can moderate. Changes are signed gossip messages (`GroupPolicyUpdate`) verified by secp256k1 signature.

4.8 Privacy & Data Isolation

Netfory enforces strict data isolation to protect user privacy.

Local-Only Storage:

- All private keys, seeds, and encrypted vaults exist only on the user's device.
- No data is transmitted to external servers unless explicitly initiated by the user (e.g., publishing a dApp, sending a message).

Full Identity Reset:

- The `delete_vault` function securely erases the encrypted key store.
- The `wipe_all_chat` function annihilates all local chat history from the `sled` database.
- P2P subscriptions are cleared.
- This ensures complete data isolation when switching or removing an identity.

No Metadata Collection:

- The tracker sees only public addresses and content hashes — not message contents, not private data.
- The P2P transport (Iroh QUIC) does not log metadata.
- There is no central server to subpoena.

4.9 Summary

The identity layer of Netfory achieves:

- **Self-sovereignty.** Users own their keys, names, and data.
- **Censorship-resistance.** No authority can revoke identity or names.
- **Privacy by design.** E2EE messaging, local-only storage, zero metadata collection.
- **Economic integration.** Paid subscriptions, paid messages, and name registration create a self-sustaining economy.
- **Human-readable addressing.** `u://` and `.sth` make the network accessible to non-technical users.

Identity in Netfory is not a service provided by a company. It is a cryptographic fact anchored on a blockchain.

---

5. The Integrated Developer Ecosystem

5.1 The Problem with Centralized CDNs

Modern web development relies heavily on centralized Content Delivery Networks (CDNs). A typical single-page application (SPA) built with React, Vue, or Bootstrap fetches dozens of JavaScript libraries, fonts, and stylesheets from services like cdnjs.cloudflare.com, unpkg.com, or jsdelivr.net. This creates three critical vulnerabilities:

1. Single Point of Failure. If the CDN goes offline, is blocked by a national firewall, or is deplatformed by its provider, the application ceases to function. The frontend cannot load. The smart contract remains accessible, but the door to it is locked.

2. Supply Chain Attacks. Centralized CDNs are high-value targets for attackers. A compromised CDN can inject malicious code into millions of applications simultaneously. The 2021 cdnjs breach demonstrated this risk: attackers gained write access to popular libraries, potentially affecting thousands of production websites.

3. Clearnet Dependency. Applications that depend on external CDNs cannot function in isolated networks (LAN, offline environments, regions with internet blackouts). The application is not truly decentralized if it requires a connection to aws.amazon.com to load its dependencies.

Netfory eliminates all three vulnerabilities through a fully decentralized, content-addressed dependency system integrated directly into the P2P network.

5.2 The Decentralized cdnjs Equivalent

Netfory includes a complete open-source library repository — a decentralized equivalent of cdnjs — distributed natively within the P2P network. This repository contains the most popular JavaScript libraries, CSS frameworks, fonts, and utilities required for modern web development:

- **JavaScript frameworks:** React, Vue, Angular, Svelte
- **Utility libraries:** Lodash, Moment.js, Axios
- **CSS frameworks:** Bootstrap, Tailwind CSS, Bulma
- **Fonts:** Roboto, Open Sans, Inter
- **Build tools:** Babel transpiler, TypeScript compiler (runtime)

Each library version is packaged as an immutable, content-addressed blob, signed by the Netfory core team with a secp256k1 private key, and distributed via the Iroh transport layer. Developers can build fully functional SPAs without making a single HTTP request to the Clearnet.

5.3 Content-Addressed Libraries

All dependencies in the Netfory ecosystem are addressed by their BLAKE3 cryptographic hash, not by their location or filename. This is the foundation of supply chain security:

Addressing Scheme:

```
sn://lib/<library-name>/<version>/<file-path>
```

Example:

```
sn://lib/react/18.2.0/umd/react.production.min.js
sn://lib/bootstrap/5.3.0/dist/css/bootstrap.min.css
sn://lib/lodash/4.17.21/lodash.min.js
```

Resolution Process:

1. The developer includes a dependency in their `manifest.json`:

```
{
  "dependencies": {
    "react": "18.2.0",
    "bootstrap": "5.3.0"
  }
}
```

2. The Netfory client computes the BLAKE3 hash of the requested library version.
3. The client queries the P2P network (via Iroh) for peers holding the blob.
4. The blob is fetched from the nearest peer, verified against the BLAKE3 hash, and cached locally.
5. The library is injected into the dApp's execution environment.

Immutability:
Once a library version is published to the network, it cannot be altered. The BLAKE3 hash is a cryptographic commitment to the exact byte-for-byte content. If an attacker attempts to publish a malicious version under the same name and version number, the hash will not match, and the client will reject it.

Cryptographic Verification:
Each library package includes a `signature.json` file, signed by the Netfory core team:

```
{
  "library": "react",
  "version": "18.2.0",
  "hash": "blake3_hex...",
  "signature": "ecdsa_compact_hex...",
  "publicKey": "compressed_pubkey_hex..."
}
```

The client verifies the signature before loading the library. This ensures that the library has not been tampered with and originates from a trusted source.

5.4 Offline-First Architecture

The decentralized CDN enables a fully offline-first development model. Once a developer has built and published a dApp, it can function indefinitely without any connection to the Clearnet.

Local Caching:

- All dependencies are cached locally in the user's `data/` directory.
- The cache is content-addressed: files are stored by their BLAKE3 hash, preventing duplication and ensuring integrity.
- When a dApp is launched, the client first checks the local cache. If all dependencies are present, the application loads instantly without querying the network.

Background Synchronization:

- If a newer version of a library is available (e.g., a security patch), the client downloads it in the background via P2P.
- The dApp continues to function with the cached version until the user explicitly updates or the cache is invalidated.

Zero Clearnet Requests:

- A Netfory dApp never makes HTTP requests to external servers. All dependencies are fetched via the `sn://` protocol from P2P peers.
- This eliminates the risk of CDN outages, supply chain attacks, and network censorship.

5.5 LAN-First Dependency Resolution

In isolated network environments (LAN, offline regions, internet blackouts), the decentralized CDN continues to function through mDNS peer discovery.

Scenario: Regional Internet Blackout
A city loses connectivity to the global internet due to a fiber cut or government-imposed TSPU filtering.

1. Global P2P discovery (n0-discovery) fails.
2. Netfory clients automatically fall back to multicast DNS (mDNS).
3. Devices on the local network broadcast their available libraries via mDNS.
4. A developer working on a dApp discovers that a peer on the same LAN has `react@18.2.0` cached.
5. The client establishes a direct QUIC connection over the LAN and fetches the library.
6. The dApp builds and runs successfully, despite the complete absence of external internet.

This is impossible with centralized CDNs. A Clearnet-based application cannot load its dependencies if the CDN is unreachable. A Netfory application can load dependencies from any peer in the local network, ensuring continuity of development and operation.

5.6 SmartNet App SDK

To simplify dApp development, Netfory provides a native SDK that injects blockchain and P2P functionality directly into the dApp's execution environment. The SDK is available as a set of Vue 3 composables and React hooks, accessible via the `window.smartholdem` bridge.

Core Hooks:

1. `useSmartNetWallet()`
Access the user's wallet and request cryptographic signatures:

```
import { useSmartNetWallet } from '@netfory/sdk';

const { address, signMessage, sendTransaction } = useSmartNetWallet();

// Request a cryptographic signature for authentication
const signature = await signMessage("Login to My dApp");

// Send a transaction
const txId = await sendTransaction({
  to: "S...",
  amount: 10,
  vendorField: "Payment for service"
});
```

2. `usePayments()`

Accept payments in STH directly within the dApp:

```
import { usePayments } from '@netfory/sdk';

const { requestPayment, getBalance } = usePayments();

// Request a one-time payment
const txId = await requestPayment({
  amount: 5,
  description: "Premium feature unlock"
});

// Check user's balance
const balance = await getBalance();
```

3. `useStorage()`

Store and retrieve data from the decentralized P2P network:

```
import { useStorage } from '@netfory/sdk';

const { upload, download, list } = useStorage();

// Upload a file to the P2P network
const cid = await upload({
  file: myFile,
  name: "document.pdf",
  encrypted: true
});

// Download a file by CID
const file = await download(cid);

// List user's uploaded files
const files = await list();
```

Context Isolation:

The SDK operates through the `window.smartholdem` bridge, which is injected by the Tauri core into the WebView. All cryptographic operations (signing, encryption) are executed in the main Rust process, never in the JavaScript context. The dApp never has direct access to the user's private key or mnemonic.

API gateway: `netfory-provider`. For dApps that need to call external clearnet APIs (e.g., price feeds, OAuth providers, third-party services), the `netfory-provider` component acts as a P2P-to-clearnet bridge. It translates `api://<nodeId>/<path>` requests into standard HTTPS calls and signs responses with an Ed25519 key, enabling clients to cryptographically verify that the response was not tampered with in transit. This preserves the zero-trust model even when dApps interact with legacy Web 2.0 services.

5.7 Developer Monetization Hooks

The SDK includes built-in hooks for monetization, enabling developers to create paid dApps, in-app purchases, and subscription services without integrating third-party payment processors.

1. Paid dApps:
A developer can set a price for their dApp. When a user attempts to install it, the client prompts for payment in STH. The payment is an on-chain transaction directly to the developer's address.

```
// In manifest.json
{
  "name": "Premium Poker",
  "price": 10,
  "currency": "STH"
}
```

2. In-App Purchases:
Developers can sell virtual goods, premium features, or digital content within their dApp:

```
const { purchaseItem } = usePayments();

// Sell a virtual item
const txId = await purchaseItem({
  itemId: "golden-sword",
  price: 2,
  description: "Legendary weapon"
});
```

3. Subscriptions:
Developers can offer recurring access to their dApp or content:

```
const { subscribe } = usePayments();

// Monthly subscription
const txId = await subscribe({
  plan: "premium",
  amount: 5,
  interval: "monthly"
});
```

Revenue Flow:
All payments flow directly from the user to the developer via on-chain transactions. There is no platform intermediary, no 30% app store cut, no payment processor fees. The only cost is the standard SmartHoldem network fee (typically <0.01 STH).

5.8 Backend Networking for dApp Authors & CORS

SmartNet dApps execute inside a native WebView with origin `sth://<app_id>/`. This is a **custom URL scheme**, not HTTP/HTTPS — and modern WebView engines (WebView2, WKWebView, WebKitGTK) enforce stricter network policies on such origins:

- **CORS:** The browser sends a preflight `OPTIONS` request. If the server does not respond with a compatible `Access-Control-Allow-Origin` header, the request is blocked.
- **Private Network Access (PNA):** Requests to private addresses (`192.168.x.x`, `10.x.x.x`, `127.0.0.1`)

from a non-secure origin require `Access-Control-Allow-Private-Network: true` in the preflight response.
- **Mixed content:** `http://...` requests from a "secure-like" origin are blocked.

The SmartNet client **intentionally does not proxy HTTP requests** from dApps. A native proxy would create a new attack surface (data exfiltration, LAN reconnaissance, client-as-relay abuse). Instead, the client relies on standard browser policies plus a correctly configured backend.

5.8.1 Three Recommended Backend Architectures

Option A — Public HTTPS backend with CORS (simplest).
Host your backend at a public domain (`https://api.mydapp.com`) with a valid Let's Encrypt certificate and correct CORS headers. Standard `fetch('https://api.mydapp.com/...')` works identically in browsers and in the SmartNet client.

Option B — Netfory-provider gateway.
Route requests through the shared `netfory-provider` gateway (`https://gw.smartholdem.io` or a self-hosted instance). The gateway translates `api://` P2P requests into clearnet HTTP and signs responses with an Ed25519 key, protecting against data substitution. No public domain required; scales horizontally.

Option C — Local development only.
For local development (`http://localhost:5173`), wrap the dev server in HTTPS via `mkcert` + Caddy/Vite. The origin becomes `https://localhost:5173`, PNA is auto-granted, and CORS works normally.

5.8.2 Minimal CORS + PNA Middleware

Backend servers must include specific headers to ensure seamless communication. Below are minimal configurations for popular frameworks.

FastAPI (Python):

```python
from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # or specific: ["sth://<your-app-id>", "http://localhost:5173"]
    allow_credentials=False,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.middleware("http")
async def pna_headers(request: Request, call_next):
    resp = await call_next(request)
    resp.headers["Access-Control-Allow-Private-Network"] = "true"
    return resp
```

Express (Node.js):

```
import express from 'express';
import cors from 'cors';

const app = express();
app.use(cors({ origin: '*', methods: '*', allowedHeaders: '*' }));

app.use((req, res, next) => {
  res.set('Access-Control-Allow-Private-Network', 'true');
  next();
});
```

Caddy (auto-HTTPS in 3 lines):

```
api.mydapp.com {
  reverse_proxy localhost:8022
}
```

5.9 WebSocket & socket.io over P2P

Beyond standard HTTP, the SmartNet client tunnels WebSocket connections through the Iroh/QUIC transport layer directly to any `netfory-provider`. This enables dApps (e.g., poker tables, live chats, real-time feeds) to maintain full-duplex channels over P2P — without requiring a public domain or TLS certificates.

5.9.1 Canonical URL Format & Invocation

Canonical URL format:

```
ws://<64-hex-nodeId>/<app-path>
```

The `<64-hex-nodeId>` is the Iroh NodeID of the provider (exactly 64 hexadecimal characters) — the same ID visible in `api://` requests and the System Console. The `<app-path>` is the standard HTTP path on the provider's side (e.g., `/pokersth/api/socket.io/`).

Correct `socket.io` invocation:

```
const nodeId = 'ab12cd34...ef'; // 64 hex characters, the provider's Iroh NodeID
const socket = io(`ws://${nodeId}`, {
  path: '/pokersth/api/socket.io',
  transports: ['websocket'],
});
```

Why this works: `socket.io-client` places the first argument into the `host` field of the final URL. A 64-hex string is a valid host, so the library accepts it. The `path` option is appended separately, preventing namespace collisions. On the wire, this results in a clean `ws://<nodeId>/pokersth/api/socket.io/?EIO=4&transport=websocket`.

5.9.2 Common Mistakes & Rescue Fallback

- ❌ `io('ws://api/pokersth', { path: '/api/socket.io' })` — The word `api` is parsed as the host, and the NodeID is lost.
- ❌ `io('api://pokersth/...')` — `socket.io` parses custom schemes incorrectly.
- ❌ `io('ws://<nodeId>/pokersth/api', { transports: ['websocket'] })` — `socket.io` treats the pathname as a namespace (e.g., `/pokersth/api`), causing the server to respond with `Invalid namespace` if it only serves the default `/` namespace.

Rescue fallback (client ≥ 1.55.35):
If a dApp uses a malformed `ws://` URL but has previously made successful `api://` fetches to the same provider, the SmartNet client attempts to recover the NodeID from its internal provider map. The connection will be "rescued," and the System Console will log: `[sth://ws] rescue ws://api/... → ws://<nodeId>/...`.
Note: This is a safety net, not a contract. It only works within the SmartNet client after a prior HTTP connection. Authors should always use the canonical format.

5.9.3 Rules for dApp Authors

1. Always keep the provider's NodeID available (via `api://` discovery, the `dev://` blockchain resolver, or as a hardcoded constant).
2. Pass the NodeID to `io()` as the host of the first argument, never as part of the `path`.
3. Use `transports: ['websocket']` unless HTTP long-polling fallback is explicitly required.
4. Verify the connection in the System Console (WS tab): it should show `open`, not `rescue` or `passthrough native WS`.
5. Never rely on the client to bypass CORS. Configure your backend correctly once, and it will work universally.

5.10 Summary

The integrated developer ecosystem of Netfory achieves:

- **Supply chain security.** All dependencies are content-addressed and cryptographically signed. Supply chain attacks are mathematically impossible.
- **Offline-first operation.** dApps function indefinitely without Clearnet connectivity. Dependencies are cached locally and synchronized via P2P.
- **LAN-first resilience.** In isolated networks, dependencies are resolved via mDNS peer discovery, ensuring continuity during internet blackouts.
- **Native monetization.** Developers can create paid dApps, in-app purchases, and subscriptions without third-party payment processors.
- **Zero platform tax.** Revenue flows directly from users to developers. There is no intermediary extraction.

The Netfory developer ecosystem transforms the web from a collection of centralized services into a decentralized, self-sustaining application layer. Developers build once, publish forever, and monetize directly. Users install applications that are immortal, censorship-resistant, and economically aligned with their creators.

6. DevHub: The Decentralized Source Code Layer

6.1 The Problem with Centralized Code Hosting

Modern software development is dominated by centralized platforms: GitHub, GitLab, Bitbucket. These services provide convenience, but they introduce a critical vulnerability: the developer's code exists at the mercy of a corporation.

A repository can be deleted due to a DMCA complaint. An account can be banned based on geographic location. A platform can be blocked by a national firewall. In each case, the developer loses access to their own work — sometimes permanently. The source code, the history of commits, the issue tracker, and the contributor community all vanish when the central server decides so.

Git was originally designed by Linus Torvalds as a distributed version control system. The protocol itself does not require a central server. The centralization is an artifact of convenience, not a technical necessity. Netfory removes the intermediary entirely.

6.2 The dev:// Protocol

DevHub introduces a native URI scheme for decentralized source code repositories:

Path	Purpose	Example
`dev://`	DevHub main hub (search, manage)	`dev://`
`dev://<username>`	Author profile (list of repos)	`dev://technolog`
`dev://<username>/<repo>`	Repository root	`dev://technolog/smart-swarm`
`dev://<username>/<repo>/issues`	Issue tracker	`dev://technolog/smart-swarm/issues`
`dev://<username>/<repo>/wiki`	Project wiki	`dev://technolog/smart-swarm/wiki`

Each repository is identified by the tuple `(username, repo_name)`. The username is resolved via the on-chain `u://` identity system. The repository content is addressed by its cryptographic hash, not by its location. Resolution is performed through the Mainline DHT, which stores branch pointers updated by the repository owner.

6.3 Git Remote Helper: The Bridge

To preserve developer workflow, Netfory implements a **Git Remote Helper** — a lightweight Rust utility named `git-remote-dev`. Git's architecture natively supports custom protocols through a well-defined helper interface. When a user executes a command with an unknown URL scheme (e.g., `dev://`), Git automatically invokes the corresponding helper binary.

Installation:

```
# Linux / macOS
install -m 0755 target/release/git-remote-dev ~/.local/bin/git-remote-dev

# Windows (PowerShell)
copy target\release\git-remote-dev.exe C:\Users\<you>\bin\git-remote-dev.exe
```

Discovery:
The helper communicates with the running SmartNet client via a loopback IPC channel. Connection parameters are stored in a discovery frame:

OS	Path
Linux / macOS	`~/.config/smartnet/devhub-ipc.json`
Windows	`%APPDATA%\smartnet\devhub-ipc.json`

Content:

```
{
  "port": "<loopback_port>",
  "token": "<random_token>"
}
```

The token protects the IPC channel from unauthorized local processes. The server listens exclusively on `127.0.0.1`.

Workflow:
The developer uses standard Git commands without modification:

```
# Clone from P2P network
git clone dev://technolog/smart-swarm

# Standard local work
git add .
git commit -m "feat: add async timeouts"

# Push to P2P network
git push origin main
```

No new commands. No new syntax. No new mental model. The helper translates standard Git operations into P2P protocol actions transparently.

6.3.1 IPC Protocol & Anti-Rollback Guarantees

The helper communicates with the running SmartNet client via a loopback HTTP channel. The discovery frame (`devhub-ipc.json`) contains a random token that authenticates the helper to the client — preventing unauthorized local processes from injecting commits.

Protocol commands (helper → client):

Git command	Helper action
`capabilities`	Declares `fetch` and `push` support
`list` / `list for-push`	`GET /refs` → list of `<sha> <ref>` + `HEAD`
`fetch <sha> <ref>`	`GET /bundle` → `git bundle unbundle` into local ODB
`push <src>:<dst>`	`git bundle create` → `POST /push` (client signs `seq+1` + DHT announce)

Anti-rollback via monotonic sequence. Every `push` increments a signed `seq` counter stored in the DHT branch pointer. The network rejects any attempt to publish an older version of the repository. This prevents malicious actors from reverting a repository to a compromised state — even if they control a majority of DHT nodes, the cryptographic signature on `seq` is authoritative.

No persistent daemon. The helper is not a background service. It launches only when invoked by Git, executes the operation in milliseconds, and terminates. There is no memory overhead, no CPU drain, no port exposure beyond the loopback channel.

IDE integration. Because the helper implements the standard `gitremote-helpers(7)` interface, all major IDEs (VS Code, WebStorm, IntelliJ, GitKraken) work without modification. The "Commit" and "Push" buttons perform exactly what they always have — the only difference is the destination.

Current limitations.

- `git push :branch` (branch deletion) is not yet supported.
- Cloning third-party repositories requires prior P2P synchronization (swarm loading of bundles via `dev_infohash` — roadmap); user's own repositories clone and push today.
- Repositories created via UI (README.md + LICENSE.md only) contain no git history; the first `git push` from a local copy creates it.

6.4 Under the Hood: DHT, Iroh, and Swarm Loading

Cloning a repository:

1. The helper intercepts the `dev://technolog/smart-swarm` URL.
2. It resolves the username `technolog` via the on-chain identity registry.
3. It queries the Mainline DHT for peers holding the repository's `bundle_hash`.
4. It initiates a swarm download via Iroh QUIC, using a Multi-Armed Bandit algorithm to optimize peer selection based on latency, bandwidth, and reliability.
5. Git objects are reconstructed locally into a standard `.git` directory.

Pushing changes:

1. The helper receives the new commits from the local Git repository.
2. It packages the delta into an encrypted Iroh collection.
3. It signs the collection with the developer's secp256k1 private key.
4. It increments a monotonically increasing `seq` counter (anti-rollback protection).
5. It updates the branch pointer in the Mainline DHT with the new `bundle_hash` and `seq`.

6. The network begins redistributing the updated repository.

Anti-rollback protection:
Each push increments a signed sequence number. The network rejects any attempt to publish an older version of the repository. This prevents malicious actors from reverting a repository to a compromised state.

Swarm loading: Multi-Armed Bandit with persistent memory.

When multiple peers hold the repository (or dApp blob), the client uses a **Multi-Armed Bandit** algorithm to dynamically select the optimal set of seeders. The algorithm balances **exploration** (probing new peers to discover fast ones) and **exploitation** (preferring known-fast peers), maximizing download speed while maintaining resilience against peer churn.

Algorithm: ε-greedy with EWMA.

- With probability $(1 - \varepsilon)$ (default $\varepsilon = 0.15$), the client performs **exploitation**: it selects the peer with the highest measured speed S_i and zero recent errors E_i.
- With probability ε, it performs **exploration**: it selects a random peer to probe its capacity.
- Speed is tracked as an **Exponentially Weighted Moving Average** (EWMA) with smoothing factor $\alpha = 0.2$, giving recent measurements more weight while preserving history.

Lock-free implementation. The bandit state (`SmartSwarmBandit`) is implemented in Rust using `std::sync::atomic` types. Peer registration uses a brief `RwLock`, but weight updates (the hot path) are pure atomic operations — guaranteeing zero contention under high load.

Persistent memory across sessions. Bandit weights are serialized to the local `sled` database under the key `swarm:weights`. On client restart, the seeder reputation is restored — fast peers from previous sessions are tried first, eliminating cold-start latency.

Failover behavior. Each download attempt has an 8-second timeout. If a peer stalls or errors, the bandit immediately penalizes it (increments E_i) and switches to the next-best candidate. Typical downloads complete 6–16 attempts before success; the user perceives a seamless 1–2 second start.

Safety guarantee. If the swarm layer encounters any unexpected error, it falls back to the legacy single-peer download path. Installation never breaks — the swarm is purely additive.

6.4.1 Swarm Loading for dApps

The same bandit engine powers dApp installation and updates (not only DevHub repositories). When a user installs a dApp, `download_blob_bandit()` iterates through all known providers (the announcing peer + headless seeders from the Network Map + tracker-discovered peers), downloads the content-addressed blob, and updates peer weights in real time.

Important design note. The current distribution format is a single blob per application. Per-file chunked distribution (iroh collections / HashSeq) is intentionally deferred — the built-in torrent engine already handles chunk-level parallelism, and changing the blob format would risk breaking deterministic hash verification, `verify_app`, deduplication, and tracker announcements for marginal gain. The swarm layer operates as **intelligent failover + best-peer selection at the blob level**.

6.5 IDE Compatibility: Zero Learning Curve

Because DevHub operates through the standard Git Remote Helper interface, all major IDEs work without modification:

- **VS Code** — native Git integration
- **WebStorm / IntelliJ** — native Git integration
- **GitKraken** — native Git integration
- **Command-line Git** — native operation

The "Commit" and "Push" buttons in the IDE interface perform exactly what they always have. The only difference is the destination: instead of sending data to GitHub's servers, the data is distributed across the P2P network.

No heavy daemons:
The helper is not a persistent background service. It launches only when invoked by Git, executes the operation in milliseconds, and terminates. There is no memory overhead, no CPU drain, no port exposure.

6.6 Issues, Wikis, and Distributed Logs

DevHub extends beyond source code. Issues, comments, and wikis are synchronized as **distributed logs** stored in the local `sled` database.

Architecture:

- Each issue, comment, or wiki page is a signed entry in a distributed log.
- Entries are replicated via P2P gossip among repository contributors.
- Conflicts are resolved using last-writer-wins semantics with cryptographic timestamps.
- The log is append-only; deletions are marked as tombstones but never physically removed.

Properties:

- **Decentralized.** No central server stores the issue tracker. Each contributor holds a full copy.
- **Censorship-resistant.** No platform can delete an issue or ban a contributor.
- **Offline-capable.** Contributors can create and resolve issues without internet connectivity. Changes synchronize when the network reconnects.

6.7 "One Peer = Immortality" Principle

DevHub operates on a fundamental guarantee: **a repository exists as long as at least one peer in the world holds a copy.**

This is the **One Peer = Immortality** principle. Unlike centralized hosting, where the deletion of a server means the permanent loss of data, DevHub distributes copies across an unbounded number of peers. Even if the original author goes offline, even if all major seeders are simultaneously deplatformed, a single residential node holding the repository ensures its survival.

Implications:

- **No single point of failure.** There is no server to seize, no domain to revoke, no account to ban.
- **Permanent availability.** Code published once remains accessible indefinitely.
- **Resistance to censorship.** Governments cannot block access to a repository that exists on thousands of independent nodes.
- **Developer sovereignty.** The author retains full control. No corporation can dictate terms, impose restrictions, or delete content.

6.8 Summary

DevHub achieves:

- **Zero learning curve.** Standard Git commands work natively. IDEs require no modification.
- **Censorship-resistance.** No central server to seize. No account to ban. No domain to revoke.
- **Permanent availability.** The repository lives as long as one peer holds it.
- **Cryptographic integrity.** All commits are signed with secp256k1. Anti-rollback protection via monotonic sequence counters.
- **Distributed collaboration.** Issues, wikis, and comments synchronize as distributed logs.
- **Optimized performance.** Multi-Armed Bandit swarm loading maximizes download speed.

DevHub transforms source code hosting from a centralized service into a decentralized protocol. Developers retain full sovereignty over their work. The code is immortal.

---

7. AI Agents & Localization Layer

7.1 Redefining "Agents" in Web 4.0

The prevailing industry narrative around Web 4.0 describes it as a network of "autonomous AI agents" that manage routing, negotiate resources, and orchestrate services on behalf of users. This framing contains a critical architectural error: it implies the existence of server-based manager agents operating as intermediaries between users and the network.

Netfory rejects this model entirely. In the Netfory architecture, **there are no server-based manager agents**. There are no cloud-hosted orchestrators routing traffic. There are no centralized AI services negotiating on the user's behalf.

Instead, **the network IS the agents**. Every user's client is itself an autonomous node that:

1. **Autonomously discovers peers** via mDNS (locally) and n0-discovery (globally).
2. **Self-verifies content integrity** via BLAKE3 hash trees — no third-party validator required.
3. **Negotiates routes directly** with other peer-agents via Iroh QUIC + Gossipsub.
4. **Self-sustains the economy** via signed heartbeats → trackers → STH rewards.

The user's device is the agent. The agent is not a service; it is the client itself. This distinction is not semantic — it is the fundamental architectural boundary between Web 4.0 and a new generation of centralized "AI middleware" masquerading as decentralization.

All intelligent behavior in Netfory — routing, verification, translation, discovery, economic negotiation — executes locally on the user's hardware. No external API calls. No cloud dependencies. No telemetry. No third-party trust.

7.2 The Localization Problem in Decentralized Networks

Decentralized networks inherit a critical limitation from their censorship-resistant design: content is immutable. A dApp published in Mandarin is readable only to Mandarin speakers. A post written in Portuguese reaches only Portuguese readers. A repository documented in Russian is opaque to English-speaking contributors.

Centralized platforms solve this through cloud-based translation APIs (Google Translate, DeepL). These services introduce three fatal vulnerabilities:

1. **Privacy leakage.** Every translated string is transmitted to a third-party server. The translator sees every message, every post, every dApp description. Metadata accumulates.
2. **Centralized dependency.** If the translation API is blocked, rate-limited, or deplatformed, the network's cross-lingual functionality collapses.
3. **Content mutation.** Centralized translators may alter, censor, or refuse to translate certain content based on corporate policy or jurisdictional law.

In a censorship-resistant network, reliance on external translation services is an architectural contradiction. The solution must be native, local, and offline-capable.

7.3 Built-in Neural Translation Layer

Netfory embeds a neural translation model directly into the desktop client. The model runs entirely on the user's hardware — no text is transmitted to external servers for translation. No API keys. No cloud endpoints. No telemetry.

Scope of translation:

- Chat messages (personal and group)
- dApp descriptions and metadata
- Post content and comments
- Repository documentation (DevHub)
- File names and tags

Translation behavior:

- Every piece of content retrieved from the P2P network is automatically translated to the user's preferred language at the presentation layer.
- The original content remains **immutable** in the P2P blob. Only the display layer adapts.
- Users can toggle between the original language and the translated version at any time.
- Translation quality is sufficient for comprehension; it is not a substitute for professional localization, but it eliminates the zero-access barrier.

7.4 Architecture: Immutable Content, Adaptive Presentation

The translation layer enforces a strict separation between **content storage** and **content presentation**.

Content Storage (Immutable):

- All content is stored in the P2P network as BLAKE3-addressed blobs.
- The original language of the content is preserved byte-for-byte.
- No translation artifacts are written back to the network.
- The author's intent, including language choice, is cryptographically protected by the author's signature.

Content Presentation (Adaptive):

- When the client retrieves a blob, the translation layer intercepts the text payload before rendering.
- The local neural model translates the text to the user's configured language.
- The translated text is rendered in the UI alongside the original (or replaces it, based on user preference).
- Translation results are cached locally to avoid redundant computation.

This architecture ensures that:

1. **The network remains language-agnostic.** Content is neither privileged nor penalized based on its original language.
2. **The author retains full control.** No third party can modify, censor, or "correct" the original text.
3. **The user gains universal access.** Language barriers dissolve without compromising the integrity of the source material.

7.5 Technical Implementation

The translation subsystem is implemented as a Rust-native module within the Tauri core, invoked from the Vue 3 frontend via the standard IPC bridge.

Model characteristics:

- Quantized neural translation model (INT8 or INT4 precision) optimized for CPU inference.
- Supports bidirectional translation among major languages (English, Russian, Mandarin, Spanish, Portuguese, Arabic, Hindi, Japanese, Korean, French, German, and others).
- Inference latency: <200 ms per message on modern consumer hardware.
- Memory footprint: <500 MB RAM during active translation.

Execution flow:

1. The frontend requests translation of a text payload via `window.smartholdem.translate(text, targetLang)`.
2. The Tauri core routes the request to the translation module.
3. The module checks the local cache (keyed by BLAKE3 hash of input + target language).
4. On cache miss, the model performs inference and returns the translated text.
5. The result is cached and returned to the frontend.

Context isolation:

- The translation model has no network access. It cannot exfiltrate data.
- The model has no filesystem access beyond its own model weights.
- All inference occurs in a sandboxed thread with no access to the user's private keys or mnemonic.

7.6 Privacy & Offline Operation

The localization layer operates under the same privacy guarantees as the rest of the Netfory stack.

Zero external communication:

- No translation request ever leaves the user's device.
- No telemetry is collected about what content is translated.
- No language preference data is transmitted to any server.

Offline capability:

- The translation model is bundled with the client binary.
- Translation functions identically in LAN-only mode, during internet blackouts, or on air-gapped machines.
- This is critical for the LAN-first autonomy property: if a region is disconnected from the global internet, users can still read and translate content shared by local peers.

No cloud fallback:

- Unlike centralized platforms, Netfory does not fall back to cloud translation when the local model is unavailable. There is no cloud to fall back to. The local model is the only translator.

7.7 Global Participation from Day One

The built-in translation layer achieves a property that no existing decentralized network provides: **true global participation from day one**.

In IPFS, Tor, I2P, and Freenet, content is accessible only to those who understand its original language. The network's growth is constrained by linguistic fragmentation. A Russian-speaking contributor cannot meaningfully participate in an English-dominant repository. A Mandarin-speaking user cannot discover a Spanish-language dApp.

Netfory eliminates this constraint at the protocol level. Every user, regardless of native language, can:

- Read and respond to any chat message.
- Understand any dApp description.
- Contribute to any DevHub repository.
- Discover any file or album in the network.

This is not a feature — it is a structural prerequisite for a truly global, censorship-resistant network. Without native localization, decentralization remains a privilege of the linguistically dominant.

7.8 Summary

The AI Agents & Localization Layer of Netfory achieves:

- **Architectural clarity.** Agents are the user's clients, not server-based intermediaries. The network IS the agents.
- **Native translation.** A quantized neural model runs locally, translating content at the presentation layer without modifying the immutable P2P source.

- **Privacy by design.** No text leaves the user's device. No telemetry. No cloud dependencies.
- **Offline capability.** Translation functions identically in LAN-only mode, during internet blackouts, or on air-gapped machines.
- **Global participation.** Linguistic barriers dissolve at the protocol level, enabling true cross-lingual collaboration from day one.
- **Author sovereignty.** The original content remains byte-for-byte immutable. Only the display layer adapts.

The localization layer is not an add-on. It is a foundational component of the Web 4.0 stack — as essential as the transport layer, the identity system, and the economic model. Without it, decentralization remains parochial. With it, the network becomes genuinely global.

---

8. Proof-of-Utility & Economics

8.1 The Economic Foundation: Money as Anti-Spam and Motivation

A decentralized network without an integrated economic layer is a tragedy of the commons. In Tor, I2P, and IPFS, all network actions are free. This produces two predictable outcomes: the network is flooded with spam and low-value content, and no participant has a financial incentive to provide long-term storage or bandwidth. The result is a network that is either unusable or unsustainable.

Netfory solves this by integrating the native token STH (SmartHoldem) directly into the protocol as both **fuel** and **anti-spam shield**. Every action that consumes network resources — registering a name, publishing an application, messaging a stranger — costs a small amount of STH. This makes spam economically devastating for attackers while simultaneously creating a perpetual revenue stream for honest participants who provide value (storage, bandwidth, content creation).

Money in Netfory is not an afterthought. It is the architectural mechanism that makes the network self-sustaining, censorship-resistant, and autonomous.

8.2 The Token: STH (SmartHoldem)

STH is the native cryptocurrency of the SmartHoldem DPoS blockchain, which serves as the trust anchor for the Netfory network.

Technical Properties:

- **Unit of account:** 1 STH = 10^8 smarthoshi (analogous to satoshis in Bitcoin). All internal sums are stored as integer smarthoshi to avoid floating-point precision errors.
- **Blockchain:** SmartHoldem DPoS — fast, low-fee transactions.
- **Cryptography:** secp256k1 (legacy bip-schnorr signatures).
- **Address format:** `S...` (base58check encoding).
- **Role in Netfory:** payment of network fees, registration of identities, publication of dApps, paid messaging, paid subscriptions, entry to paid groups, and seeder rewards.

STH is not a speculative asset detached from utility. It is the physical fuel that powers the storage and distribution of data in the real world. As the network grows, liquidity is locked or burned in reward pools, creating deflationary pressure directly proportional to network usage.

8.3 Economic Actions & Costs

The following table enumerates all protocol-level economic actions in Netfory. Every fee is visible to the user before the transaction is signed — there are no hidden charges.

Action	Cost	Purpose
Transfer STH	Dynamic network fee	Base blockchain fee
Register `u://name`	1 STH (on-chain tx)	Anti-squatting, unique human-readable identity
Publish X25519 chat key	Network fee (`CHAT_KEY_FEE_STH`)	Enable P2P messaging
Publish dApp	Dynamic by payload size (base ~100 STH)	Anti-spam for the store; free drafts allowed
Update dApp manifest	1 STH	Prevent frivolous updates
Message to stranger	`msg_cost_sth` (set by receiver)	Anti-spam + recipient monetization
Subscribe to author (`sub:<user>`)	On-chain fee	Anti-spam + creator monetization
Enter paid group	`entry_fee` (set by owner)	Community monetization

All fees are paid directly to the protocol or to the recipient. There is no platform intermediary, no 30% app store cut, no corporate extraction.

8.4 Anti-Spam Economic Proof

In legacy P2P networks, the cost of spam is zero:

$$C_{spam}^{Tor/I2P/IPFS} = 0$$

This makes the network trivially floodable. An attacker can generate millions of messages, fake identities, or junk content at no cost, degrading the experience for all honest participants.

In Netfory, every message to a stranger is an on-chain transaction with a cost set by the receiver:

$$C_{spam}^{Netfory} = N \times msg\_cost\_sth$$

Numerical example:
If $N = 1{,}000{,}000$ messages and $msg_cost_sth = 0.01$ STH:

$$C_{spam} = 1{,}000{,}000 \times 0.01 = 10{,}000 \text{ STH}$$

Burning 10,000 STH to deliver spam that will be ignored is a provably irrational expenditure. The attack is economically devastating to the attacker before a single message is delivered.

This same principle applies to:

- **Store spam:** Publishing a dApp costs ~100 STH. Mass-publishing junk applications is financially ruinous.
- **Identity squatting:** Registering a `u://name` costs 1 STH. Registering thousands of names to hoard them is economically unviable.
- **Group spam:** Entering a paid group requires `entry_fee`. Mass-joining groups to spam is costly.

The economic shield is not a policy — it is a mathematical property of the protocol.

8.5 The Earn STH Model: Proof-of-Utility

Network participants who provide storage and bandwidth are compensated through a native Proof-of-Utility model. Seeders (both headless VPS nodes and hybrid desktop clients) broadcast signed heartbeats every 5 minutes. Trackers maintain a 24-hour sliding window and compute an uptime coefficient K_{up}.

Seeder Income Formula:

$$Estimated = (Base + seededGb \times density + activeApps \times PerApp) \times K_{up}$$

Parameters:

Parameter	Typical Value	Description
`Base`	~5 STH/day	Base invisible income per active seeder
`density`	~0.5 STH/GB/day	Depends on the application's boost-pool size
`PerApp`	Bonus per distinct app	Seeding 3 dApps is more profitable than 1 dApp, regardless of size
`K_up`	0.0 – 1.0	Uptime multiplier (see §8.6)

Numerical example:
A seeder with `Base = 5`, `seededGb = 100`, `density = 0.5`, `activeApps = 10`, `PerApp = 0.2`, `K_up = 1.0`:

$$Estimated = (5 + 100 \times 0.5 + 10 \times 0.2) \times 1.0 = 57 \text{ STH/day} = 20{,}805 \text{ STH/year}$$

The more uptime a seeder maintains, and the more distinct useful applications it redistributes, the higher its income. This aligns individual profit with network health.

8.6 Availability Consensus: The Sliding Window

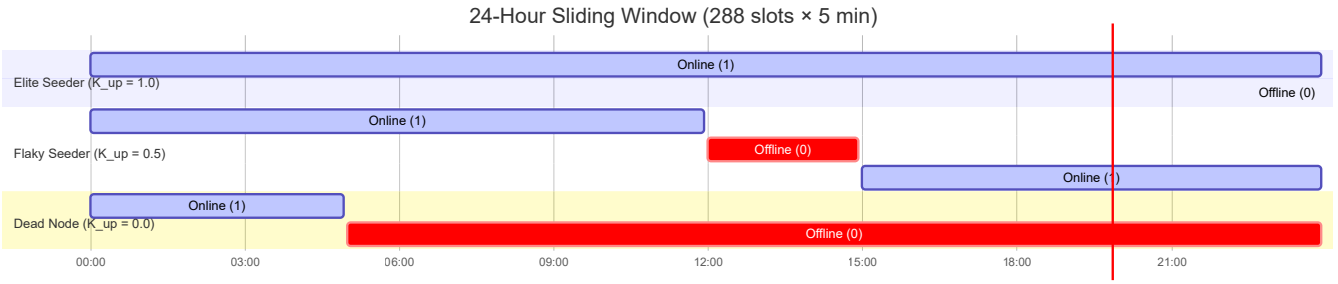
To prevent high node churn and punish "lazy seeders" who run their nodes sporadically, the tracker enforces a strict mathematical model of availability.

Signed Heartbeats:
Every 5 minutes, each seeder generates a lightweight heartbeat packet containing:

- Current timestamp
- `reward_address` (SmartHoldem address)
- List of `app_id` hashes being held

The packet is signed with the seeder's secp256k1 private key and broadcast to trackers.

24-Hour Sliding Window:
The tracker maintains a sliding window of exactly 288 time slots (12 slots/hour × 24 hours) in its `sled` database. A valid, time-aligned heartbeat writes `1` to the current slot; a missed interval writes `0`. Data is stored as a compact bitmask `[u8; 36]`.



Note: The chart above visualizes the 288-slot bitmask `[u8; 36]`. Each block represents a 5-minute heartbeat interval. Rewards are multiplied by the non-linear stability coefficient K_{up} based on the percentage of '1's.

Uptime Coefficient K_{up}:
Before the daily reward distribution, the tracker computes the node's individual uptime as the percentage of `1`s in the window. The reward is multiplied by a non-linear stability coefficient:

Uptime Range	Seeder Class	K_{up}
98.0% – 100.0%	Elite Seeder	1.0 (maximum)
90.0% – 97.9%	Flaky Seeder	0.5 (50% penalty)
< 90.0%	Dead Node	0.0 (voided)

Properties:

- **Anti-sybil:** Heartbeats are signed; a tracker cannot be fooled into attributing uptime to a fake address.
- **Graceful degradation:** A seeder with 95% uptime still earns, but at half rate — reflecting the extra mirror-searching burden imposed on the network.
- **Zero-reward floor:** Below 90% uptime, the node is classified as dead. Unclaimed coins remain in the reward pool for the next epoch.

Client-side fallback:
If a tracker returns `K_up = 0` (fresh or low-trust node), the client falls back to its own local estimate (`uptime_to_kup`) so that the income forecast does not collapse to zero prematurely.

8.7 Boost-Pools: Market-Driven Availability

Authors who require maximum download speed and redundant distribution of their dApp can activate a **Boost-Pool** — a market mechanism that directly incentivizes seeders.

Mechanism:

1. The author sends STH to the Global Seeding Pool address (`SappsPqDQXCRvoi1V2Hs9xycQeT1ptJWV1`) with a unique `vendorField` marker: `sth_boost:<app_id>`.

2. Trackers record the transaction and dynamically increase the application's `priority_weight`.
3. Seeders, observing the elevated weight, automatically prioritize the dApp for bandwidth allocation.
4. Under disk pressure, non-boosted applications are evicted first (Eviction Policy), while boosted content is retained.

Economic effect:
A market emerges: authors pay for availability, seeders earn for redistributing popular content. The tracker aggregates pool sizes via `GET /apps`, and clients use the sum to estimate `density` (STH per GB/hour) when forecasting income.

Boost-pools transform availability from a public-goods problem into a self-regulating market.

8.8 Monetization Avenues

Netfory provides multiple native monetization paths for both users and developers. All payments are peer-to-peer, on-chain, and free of platform tax.

8.8.1 For Users

1. **24/7 Seeding (Earn STH).** Keep the client online, redistribute popular dApps → stable passive STH income, scalable by the uptime multiplier and number of applications.
2. **Paid Inbox.** Set `msg_cost_sth > 0` to monetize attention. Experts, bloggers, and consultants earn directly from inbound messages.
3. **Paid Subscriptions.** Content creators charge STH for subscriptions to their channel or feed.
4. **Paid Groups.** Create closed communities with an `entry_fee` (on-chain verifiable receipt, roadmap).
5. **Names as Assets.** Register short or brandable `u://` names on a first-win basis; names become tradable digital assets.

8.8.2 For Developers

1. **Paid dApps.** Publish an application → it enters the store and P2P distribution → charge for installation or access.
2. **In-App Payments in STH.** Built-in purchases, premium features, and unlocks within the dApp (payments route through the SmartNet wallet).
3. **Subscription Services.** SaaS-style logic on STH: recurring payments for access.
4. **Boost-Economy.** Fund a boost-pool to ensure seeders actively redistribute the application → higher availability → more users.
5. **B2B Infrastructure.** Operate a self-hosted tracker (paid/premium application listings, availability analytics) or a headless-seeder farm.

8.8.3 Future Monetization Variants (Roadmap)

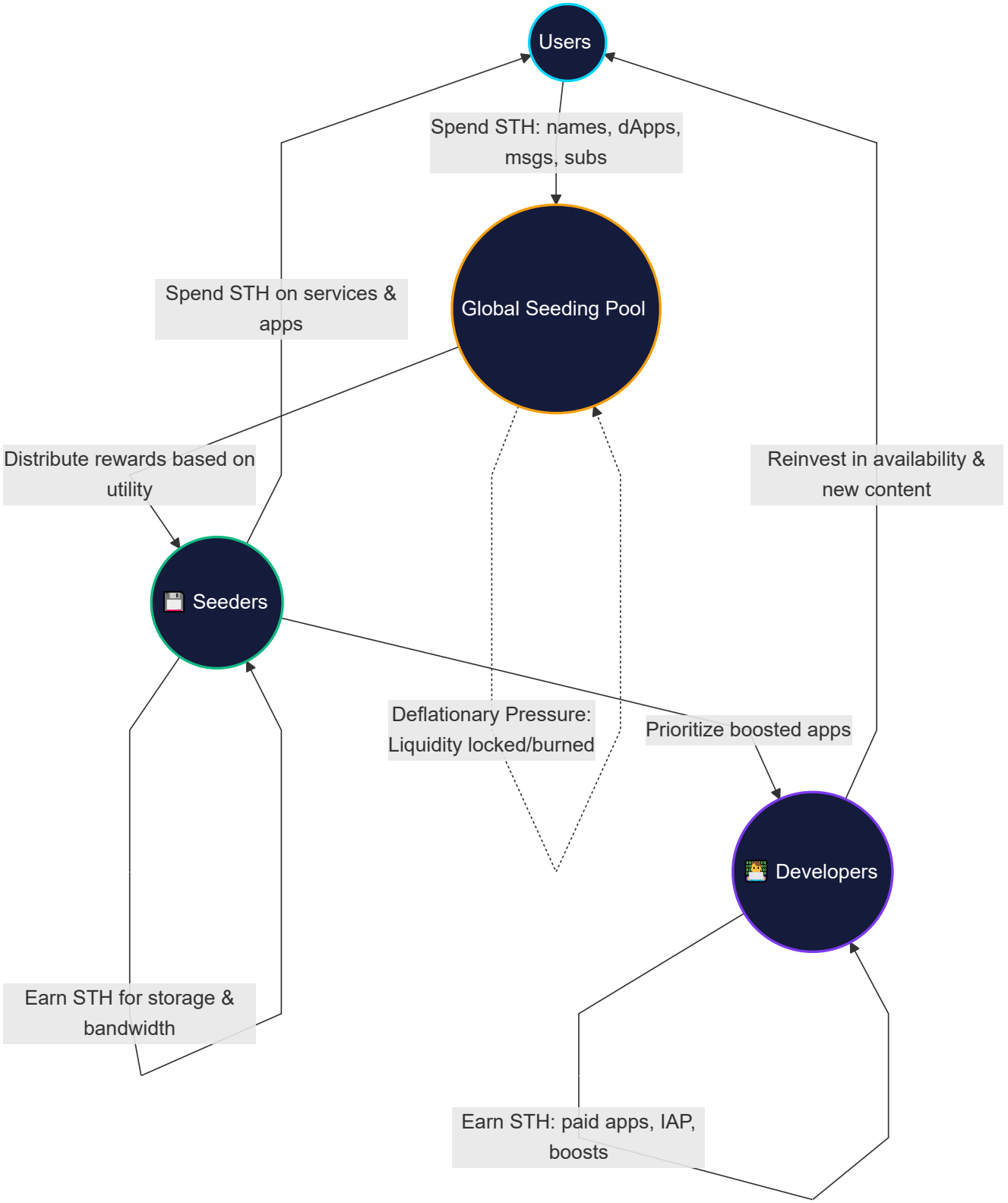
- On-chain verifiable receipts for paid group entry.
- Secondary marketplace for `u://` names (auctions).
- Paid premium listing in tracker recommendations.
- Tracking-free ads: paid promo-cards for dApps, ranked by boost-pool, with zero personal data

collection.

- Availability staking: STH collateral from seeders as an uptime guarantee → higher K_{up} and reputation.
- App SDK with built-in billing (subscriptions, IAP — see §5.7).

8.9 The Perpetual Economic Loop

Netfory closes the hosting economy into a self-sufficient cycle, creating organic, continuous demand for STH.



The Loop:

1. **Users spend STH** to register names, publish dApps, send paid messages, subscribe to creators, and enter paid groups.
2. **Fees flow to the Global Seeding Pool** and directly to content creators.
3. **Seeders earn STH** for providing storage and bandwidth, proportional to their uptime and the value of content they redistribute.
4. **Seeders spend STH** on names, dApps, and services — recycling the token.
5. **Developers earn STH** from paid dApps, in-app purchases, and boost-pools — reinvesting in availability and new content.

Key properties:

- **Spam is economically unviable.** Every "free" attack vector is paid → network quality rises.
- **Money flows to creators of value** (authors, developers, seeders), not to a platform-intermediary.
- **No single point of failure.** No "platform tax." STH moves P2P between participants; fees are network fees, not corporate revenue.
- **Deflationary pressure.** As the network grows, liquidity is proportionally locked or burned in reward pools.

This is not a tokenomics diagram drawn for investors. It is the mechanical heart of the network — the reason Netfory can operate indefinitely without venture capital, without corporate subsidies, and without centralized governance.

8.10 Token Distribution & Emission Model

Transparency in token allocation is a cornerstone of the SmartHoldem ecosystem. The initial distribution of SmartHoldem (STH) was conducted with a strong emphasis on community trust and decentralized oversight, avoiding the pitfalls of centralized pre-mines.

Initial Distribution (2017):
The genesis supply was capped at a fixed amount of **240,000,000 STH**. The distribution was executed under the strict control of well-known, community-voted escrows from the BitcoinTalk forum, ensuring fairness and preventing manipulation:

- **83%** allocated to ICO participants.
- **8%** allocated to advisors and angel investors.
- **6%** reserved for the Development Fund.
- **3%** allocated to Bounty campaign participants.

Evolution of the Emission Model:
Originally, the SmartHoldem DPoS blockchain operated under an inflationary model with a capped emission trajectory, planning to forge a maximum of 256,425,000 STH by the year 2041 to incentivize delegate validators.

However, following the launch of the global **Update 2.0** of the platform infrastructure, SmartHoldem fundamentally shifted its monetary policy. **Block emission in the DPoS consensus has been completely stopped.** The network has successfully transitioned from an inflationary model to a strictly **limited and deflationary emission model**.

Current Supply Dynamics:
At the time of writing, the total supply of SmartHoldem is **249,677,907 STH** (accounting for the forged amount prior to the emission halt). Furthermore, the protocol actively reduces supply through burning mechanisms, with **92,093 STH** already permanently burned.

Combined with the new Web 4.0 utility — where STH is locked or burned in seeding pools, boost-pools, name registrations, and anti-spam fees—this halted emission creates powerful, sustained deflationary pressure. STH is no longer just a consensus reward; it is a scarce, utility-driven asset backing a real-world decentralized infrastructure.

8.10.1 Token Velocity & Deflation Mechanisms

The economic design of Netfory ensures that STH is not a speculative asset detached from utility. Every protocol-level action either burns, locks, or recycles STH, creating sustained deflationary pressure directly proportional to network usage. The following table summarizes the primary mechanisms that govern token velocity and supply reduction:

Mechanism	Effect on STH Supply	Status
Registration of `u://` identity (1 STH)	**Burn** — permanently removed from circulation	✅ Live
Registration of `<name>.sth` domain	**Burn** — fee scaled by name length and demand	✅ Live
Publication of dApp (~100 STH)	**Lock** — transferred to Global Seeding Pool (`SappsPqDQXCRvoi1V2Hs9xycQeT1ptJWV1`)	✅ Live
dApp manifest update (1 STH)	**Burn** — prevents frivolous updates	✅ Live
Paid messages (`msg_cost_sth`)	**P2P transfer** — recipient monetization, small network fee burned	✅ Live
Paid subscriptions (`sub:<user>`)	**P2P transfer** — creator monetization, small network fee burned	✅ Live
Paid group entry (`entry_fee`)	**Lock** — held in community pool	✅ Live
Boost-Pool contributions (`sth_boost:<app_id>`)	**Lock** — staked for seeder incentives, recycled via rewards	✅ Live
DPoS block emission halt (Update 2.0)	**Hard cap** — inflationary supply permanently stopped at ~249.7M STH	✅ Live
Proof-of-Storage slashing (invalid proofs)	**Burn** — staked collateral destroyed	🛠 Roadmap
Availability Staking (uptime guarantee)	**Lock** — seeder collateral, slashable on downtime	🛠 Roadmap

Mechanism	Effect on STH Supply	Status
Secondary marketplace for `u://` names	**Burn** — royalty fee on each trade	🛠 Roadmap

Key properties:

- **No inflationary escape valve.** With block emission permanently halted, the only way new STH enters circulation is through the existing supply being unlocked from pools — and this is always offset by new burns from registrations and publications.
- **Utility-driven demand.** Every active user, developer, and seeder creates organic demand for STH. The token is not a governance token or a speculative asset; it is the physical fuel of a decentralized infrastructure.
- **Deflationary feedback loop.** As the network grows → more dApps are published → more names are registered → more STH is burned/locked → supply decreases → scarcity increases → utility-backed value rises.

This is not a tokenomics diagram drawn for investors. It is the mechanical heart of the network — the reason Netfory can operate indefinitely without venture capital, without corporate subsidies, and without centralized governance.

8.11 Summary

The Proof-of-Utility economic model of Netfory achieves:

- **Mathematical anti-spam.** $C_{spam} = N \times msg_cost_sth$ makes large-scale spam economically devastating.
- **Aligned incentives.** Seeders earn proportionally to their uptime and the distinct value they redistribute.
- **Market-driven availability.** Boost-pools transform content distribution into a self-regulating market.
- **Multi-path monetization.** Users and developers have native, peer-to-peer revenue streams without platform tax.
- **Perpetual self-sufficiency.** The economic loop closes on itself — the network funds itself.
- **Autonomy.** No venture capital, no corporate subsidies, no centralized governance required.

The economy of Netfory is not an add-on. It is the structural prerequisite for a censorship-resistant, autonomous, agent-native Web 4.0 infrastructure. Without it, decentralization is a hobby. With it, decentralization is a permanent, self-sustaining reality.

---

9. Proof-of-Storage: Cryptographic Custody Verification

9.1 The Storage Verification Problem

A decentralized storage network faces a fundamental cryptographic challenge: how to mathematically prove that a seeder node is storing a file at any given moment, without downloading the entire file for verification.

If the network simply requests "send me the file," the verification traffic would consume the entire bandwidth of the WAN. If the network only requests the file's hash, a malicious seeder can compute the hash once, store only those 32 bytes, delete the original file, and indefinitely return the correct hash while consuming zero storage. This is known as a **Generation Attack** or **Outsourcing Attack** — the seeder downloads the file from a neighbor only at the moment of verification, then discards it.

Netfory solves this through a two-layer cryptographic verification system: **Merkle spot-checking** for lightweight, instantaneous verification, and **Proof of Replication (PoRep)** for protection against Sybil attacks.

9.2 Spot-Checking via Merkle Trees

The first layer of verification forces seeders to prove possession of random micro-chunks of the file. This is computationally trivial for honest seeders and mathematically impossible to fake for dishonest ones.

Preparation (Chunking & Merkle Root):
When a dApp or file is ingested into Netfory, it is sliced into small chunks (e.g., 64 KB each). A Merkle Tree is constructed from these chunks. The root hash (Merkle Root) is permanently anchored in the blockchain or the tracker's persistent `sled` database.

Entropy Source (Blockchain as Random Number Generator):
At regular intervals (e.g., every 100 SmartHoldem blocks), the network issues a cryptographic challenge. The entropy source is the hash of the latest blockchain block — a value that cannot be predicted in advance.

Challenge Generation:
To prevent seeders from colluding, the target chunk index is calculated uniquely for each node:

$$TargetChunkIndex = Hash(Hash(Block) + SeederWalletAddress) \mod TotalChunks$$

This ensures that each seeder receives a different challenge, making coordination impossible.

Proof Generation:
The seeder receives the command: "Prove you have chunk #42." The seeder extracts the raw 64 KB data of chunk #42, combines it with the block hash, and generates a Merkle Proof — the cryptographic path from that specific chunk to the Merkle Root.

Instant Verification:
The DPoS delegates or the tracker receive this Merkle Proof. They do not need the entire file. They take the chunk data, hash it along with the provided Merkle path, and verify in <1 microsecond whether the final hash matches the stored Merkle Root.

If the hash matches, the seeder has mathematically proven possession of the file. If the seeder fails to respond within 3 blocks, it is penalized or ejected from the seeder pool.

9.3 Proof of Replication: Unique Sealing

Merkle spot-checking is elegant, but it has a vulnerability: **Sybil Attack**. If a file is popular, a single seeder can spin up 10 virtual nodes on a single physical disk, all holding the same file. All 10 nodes would pass the spot-check using the same data, claiming 10× the storage contribution while consuming only 1× the physical storage.

Netfory implements a simplified, Rust-native variant of Proof of Replication (PoRep) to eliminate this attack vector.

Unique Sealing Mechanism:
When a seeder agrees to store a file, it does not merely download a copy. It **seals** (encrypts) the file with a unique key derived from its wallet address:

$$SealedFile = File \oplus PBKDF2(WalletAddress, NetworkSalt)$$

The Merkle Root is then computed from the sealed file, not the original.

Properties:

- **Uniqueness.** The sealed file on this specific seeder's disk is cryptographically unique. A neighbor cannot provide answers to challenges because their file is sealed to a different wallet.
- **Physical storage requirement.** The seeder must allocate physical SSD space for this sealed array. There is no shortcut.
- **Sybil resistance.** Running 10 virtual nodes requires 10× the physical storage, as each node must seal the file independently.

This is a lightweight alternative to Filecoin's zk-SNARKs-based PoRep, which would melt the CPUs of budget VPS nodes. Netfory's approach is computationally trivial for honest seeders and mathematically binding for dishonest ones.

9.4 Rust Implementation: StorageChallenge

The verification logic is implemented as a native Rust module (`storage_proof.rs`) within the SmartNet core.

```rust
// Author: TechnoLOg
// Date: 2026
// Module: storage_proof.rs – Cryptographic storage challenge computation

use sha2::{Sha256, Digest};

pub struct StorageChallenge {
    pub file_merkle_root: [u8; 32],
    pub total_chunks: u64,
    pub block_entropy: [u8; 32],
}

impl StorageChallenge {
    // Determine which chunk this specific seeder must prove
    pub fn get_target_chunk(&self, seeder_address: &str) -> u64 {
        let mut hasher = Sha256::new();
        hasher.update(&self.block_entropy);
        hasher.update(seeder_address.as_bytes());
        let result = hasher.finalize();

        // Convert first 8 bytes of hash to u64 and modulo by total chunks
        let mut bytes = [0u8; 8];
        bytes.copy_from_slice(&result[0..8]);
        let random_num = u64::from_be_bytes(bytes);
        random_num % self.total_chunks
    }
```

```
    // Verify the Merkle Proof submitted by the seeder
    pub fn verify_proof(
        &self,
        chunk_data: &[u8],
        proof_path: Vec<[u8; 32]>,
        target_chunk: u64,
    ) -> bool {
        // 1. Hash the chunk itself
        let mut current_hash = Sha256::digest(chunk_data);

        // 2. Ascend the Merkle Tree using the provided proof path
        for node in proof_path {
            let mut hasher = Sha256::new();
            hasher.update(current_hash);
            hasher.update(node);
            current_hash = hasher.finalize();
        }

        // 3. If the root matches the network's stored root, the seeder is honest
        current_hash.as_slice() == self.file_merkle_root
    }
}
```

Performance characteristics:

- Challenge generation: <1 ms per seeder.
- Proof generation: <10 ms per chunk (64 KB).
- Verification: <1 microsecond per proof.
- Memory footprint: negligible (no large allocations).

This is orders of magnitude faster than zk-SNARK-based systems, making it feasible for budget VPS nodes ($2/month) to participate in the storage economy.

9.5 Integration with SmartHoldem DPoS

The Proof-of-Storage mechanism is tightly integrated with the SmartHoldem DPoS blockchain, creating a self-regulating economic ecosystem.

StorageProofTx:
Every hour, each seeder submits a lightweight system transaction (`StorageProofTx`) to the blockchain, attaching the result of the cryptographic challenge.

Delegate Validation:
When SmartHoldem delegates assemble a block, they extract the `StorageProofTx`, run it through `verify_proof()` (taking nanoseconds), and, if valid, include the node in the reward distribution list.

Automatic Clearing:
Once per day, the core distributes STH tokens to all active seeders proportionally to their proven storage volume. No manual intervention. No centralized oracle. The blockchain is the trust anchor.

Penalties:

- Failure to respond to a challenge within 3 blocks: temporary suspension from the reward pool.

- Repeated failures: permanent ejection from the seeder set.
- Invalid proofs: slashing of staked collateral (see §9.6).

9.6 Economic Incentives and Penalties

Proof-of-Storage transforms the network into a self-regulating economic ecosystem where participants are financially incentivized to provide genuine, long-term storage.

Rewards:

- Seeders who pass spot-checks receive STH proportional to their proven storage volume.
- The reward is drawn from the Global Seeding Pool (funded by dApp publication fees and boost-pool contributions).
- Higher uptime and more distinct applications → higher rewards (see §8.5 Earn STH formula).

Staking (Collateralized Availability):
Professional headless seeders who wish to claim elevated reward rates must stake STH as collateral. If a seeder fails spot-checks, the stake is partially slashed or burned. This creates a financial guarantee of availability.

Market Dynamics:

- Seeders compete to store popular, boost-pool-funded applications (higher density → higher rewards).
- Users benefit from redundant, high-speed distribution.
- The network self-optimizes: popular content is over-replicated, niche content is preserved by dedicated seeders.

Deflationary Pressure:
As the network grows, liquidity is proportionally locked or burned in reward pools. The more storage is proven, the more STH is removed from circulation, creating deflationary pressure directly correlated with network utility.

9.7 Summary

The Proof-of-Storage system of Netfory achieves:

- **Mathematical custody verification.** Seeders prove possession of random chunks via Merkle spot-checking, verified in <1 microsecond.
- **Sybil resistance.** Proof of Replication (unique sealing) ensures that each seeder must allocate physical storage, preventing virtual node duplication.
- **Lightweight computation.** No zk-SNARKs. No heavy CPU overhead. Feasible for budget VPS nodes.
- **Blockchain integration.** `StorageProofTx` validated by DPoS delegates, automatic daily clearing of rewards.
- **Economic self-regulation.** Staking, slashing, and boost-pools create a market-driven storage economy.
- **Deflationary tokenomics.** Proven storage locks or burns STH, creating deflationary pressure proportional to network growth.

The network does not trust seeders. It mathematically verifies them. Every byte of storage is proven, every reward is earned, every penalty is enforced by cryptographic consensus.

This is not a theoretical model. It is a Rust-native implementation, integrated with the SmartHoldem DPoS blockchain, and operational in the Netfory stack.

---

10. The Tracker: Coordination, Not Control

10.1 The Role of the Tracker

A common misconception about peer-to-peer networks is that they require no coordination whatsoever. In practice, pure DHT-based systems (such as vanilla BitTorrent or early IPFS) suffer from slow peer discovery, high bootstrap latency, and vulnerability to eclipse attacks. Netfory solves this through a lightweight coordination layer: the **Tracker**.

The Tracker is a thin HTTP server that performs exactly three functions:

1. **Catalog aggregation.** It maintains an in-memory registry of signed announcements from seeders, mapping `app_id` → `content_hash` → `providers`. Clients query this catalog to discover peers holding specific dApps.
2. **Uptime accounting.** It receives signed heartbeats from seeders every 5 minutes, maintains a 24-hour sliding window per address, and computes the availability coefficient K_{up} (see §8.6).
3. **Bootstrap acceleration.** It helps new or distant nodes quickly locate peers, bypassing the slow convergence of pure DHT lookups.

Critical property: The Tracker does NOT store content. It does not host blobs. It does not serve dApps. It does not see message contents. It is a coordination aid, not a trust anchor. Content integrity is verified by the client via BLAKE3 hash trees and author signatures — not by the tracker.

The network functions perfectly well without trackers via mDNS (local) and n0-discovery (global). Trackers are an optimization, not a dependency. If every tracker in the world went offline simultaneously, Netfory would continue to operate — peer discovery would simply be slower.

10.2 Architecture

The tracker is implemented as a single Rust binary (`smartnet-tracker`) built on the following stack:

Component	Technology	Purpose
HTTP framework	`axum 0.8`	Async HTTP server with routing
Async runtime	`tokio 1`	Multi-threaded event loop
Persistent storage	`sled 0.34`	Embedded key-value database
Cryptography	`secp256k1 0.29`, `sha2 0.10`, `ripemd 0.1`	ECDSA verification, hashing, address derivation
Encoding	`bs58 0.5`, `hex 0.4`	Base58Check, hex serialization
HTTP client	`ureq 2`	Blocking client for metrics scraping

Component	Technology	Purpose
Middleware	`tower-http 0.6`	CORS + tracing
Logging	`tracing 0.1`	Structured logging

Data model:

- **In-memory registry.** `HashMap<app_id, AppRecord>` and `HashMap<endpointId, Provider>`, with TTL-based expiration.
- **Persistent store.** `sled` database at `./tracker-data` (configurable via `SMARTNET_TRACKER_DB`). Survives restarts; providers are restored with a TTL grace window.
- **Pruning loop.** Every 30 seconds, a background task evicts providers with expired TTL from both memory and `sled`. When all providers of a dApp are evicted, the dApp record is deleted entirely.

Configuration:

Variable	Default	Description
`SMARTNET_TRACKER_PORT`	7374	HTTP server port
`SMARTNET_TRACKER_TTL`	120	Announcement TTL (seconds)
`SMARTNET_TRACKER_DB`	`./tracker-data`	Path to sled database
`SMARTNET_TRACKER_LOG_ANNOUNCE`	`true`	Log every announce

Build optimization:

```
[profile.release]
opt-level = "z"   # minimal binary size
lto = true        # link-time optimization
strip = true      # remove debug symbols
```

The resulting binary is a single, self-contained executable deployable on any $2/month VPS.

10.3 Trust Model: Why the Tracker Cannot Fake Content

The tracker is explicitly designed to be **untrusted for content**. This is enforced by three independent cryptographic mechanisms:

1. Content-addressed identity.
Every dApp is identified by its `app_id`, computed as:

$$app\_id = base58check(0x3F \parallel RIPEMD160(secp256k1\_compressed\_pubkey))$$

The `app_id` is a deterministic derivative of the author's public key. It cannot be forged without the corresponding private key.

2. Signed announcements.
Every `POST /announce` payload must include an ECDSA signature over `SHA-256(id | name | description | merkle)`. The tracker verifies:

- The signature is valid.
- The `app_id` matches the public key.
- The `name`, `description`, and `merkle` fields are protected by the signature.

Announcements with invalid signatures or mismatched IDs are rejected immediately.

3. Client-side verification.
When a client fetches a dApp blob from a provider discovered via the tracker, it independently verifies:

- The BLAKE3 hash of the downloaded blob matches the `hash` field in the announcement.
- The `content.json` manifest inside the tar archive contains a valid author signature.

Even if a tracker were compromised and returned malicious announcements, the client would reject any blob that does not cryptographically match the expected hash. The tracker cannot substitute content.

Summary of trust boundaries:

- The tracker is trusted for **discovery** (which peers hold which apps).
- The tracker is NOT trusted for **content** (what the apps contain).
- The tracker is NOT trusted for **identity** (who the author is — this is anchored on-chain).

10.4 HTTP API

The tracker exposes a minimal, RESTful HTTP API. All endpoints support CORS (`Access-Control-Allow-Origin: *`) to enable web-based dashboards and clients.

Method	Path	Description
`GET`	`/`	HTML dashboard (auto-refresh 3s)
`POST`	`/announce`	Heartbeat from a seeder node
`GET`	`/apps`	All known dApps with providers
`GET`	`/app/:id`	Providers of a single dApp
`GET`	`/seeds`	Unique live endpoints
`GET`	`/seeders`	Seeder leaderboard (JSON)
`GET`	`/map`	HTML Network Map dashboard
`GET`	`/stats`	Tracker statistics (JSON)
`GET`	`/health`	Liveness check

10.4.1 POST /announce

A seeder broadcasts its presence and the list of dApps it is currently seeding.

Request:

```
{
  "node": "S...",
  "endpointId": "iroh_endpoint_id",
  "metricsUrl": "http://ip:port/metrics",
  "apps": [
    {
      "id": "S...",
      "name": "My dApp",
      "description": "...",
      "tags": ["poker", "game"],
      "merkle": "sha256_hex",
      "version": 1,
      "hash": "iroh_blob_hash",
      "signature": "ecdsa_compact_hex",
      "publicKey": "compressed_pubkey_hex"
    }
  ]
}
```

Processing:

1. Each app is verified (signature + ID↔pubkey match).
2. Providers are updated by `endpointId` (deduplicated).
3. Version is updated only if the new version > current version.
4. Affected records are persisted to `sled`.
5. If `metricsUrl` is provided, the seeder is registered for Network Map scraping.

Response:

```
{ "ok": true, "accepted": 3, "rejected": 0, "ttlSecs": 120 }
```

10.4.2 GET /apps

Returns all known dApps (deduplicated), sorted by seed count (descending).

Response:

```
[
  {
    "name": "My dApp",
    "description": "...",
    "tags": ["poker"],
    "merkle": "sha256_hex",
    "version": 2,
    "seeds": 5,
```

```
    "providers": [
      {
        "endpointId": "...",
        "node": "S...",
        "hash": "iroh_blob_hash",
        "version": 2
      }
    ]
  }
]
```

10.4.3 GET /seeders

Returns the seeder leaderboard for the Network Map. Aggregates metrics scraped from all known seeders.

Scoring formula:

$$Score = GB\_seeded \times \left(\frac{uptime\_secs}{3600} \right)$$

This is the **Byte-Hours** metric — the product of storage volume and uptime duration.

Response:

```
[
  {
    "endpointId": "...",
    "node": "S...",
    "metricsUrl": "http://...",
    "appsSeeded": 12,
    "gbSeeded": 4.567,
    "uptimeSecs": 86400
  }
]
```

10.4.4 GET /stats

Returns tracker-level statistics.

Response:

```
{
  "ok": true,
  "uptimeSecs": 86400,
  "ttlSecs": 120,
  "apps": 42,
  "providers": 128,
  "endpoints": 35,
  "logAnnounce": true,
  "totals": {
    "announceRequests": 1500,
    "accepted": 4500,
    "rejected": 23
```

```
  },
  "topApps": [
    { "id": "S...", "name": "App", "version": 2, "seeds": 10 }
  ]
}
```

10.5 Multi-Tracker Fallback & Network Resilience

Netfory clients maintain a configurable list of trackers (stored locally in the Settings → Network section). The client iterates through the list with a round-robin fallback strategy.

Resilience properties:

- **Single tracker failure.** If one tracker goes offline, the client seamlessly switches to the next in the list. Peer discovery continues uninterrupted.
- **All trackers failure.** If every tracker is unreachable, the client falls back to mDNS (local) and n0-discovery (global). Discovery is slower but fully functional.
- **Tracker compromise.** If a tracker is compromised and returns malicious data, the client's BLAKE3 verification rejects any tampered content (see §10.3).

This design ensures that the tracker is a **performance optimization**, not a **single point of failure**. The network's survival does not depend on any tracker.

10.5.1 Trackerless Discovery via Mainline DHT

To eliminate the HTTP tracker as even a theoretical single point of failure, Netfory implements a **trackerless peer discovery layer** built on the BitTorrent Mainline DHT (BEP5), adapted for the Iroh transport.

The Architectural Challenge. BitTorrent's DHT resolves `infohash → IP:port`. Iroh, however, connects by `EndpointId` (a public-key cryptographic identity), not by IP. The DHT therefore answers "where" but not "who". Netfory bridges this gap with a one-RTT identity resolution step.

Mechanism.

1. **Deterministic infohash.** For each dApp, the client computes:

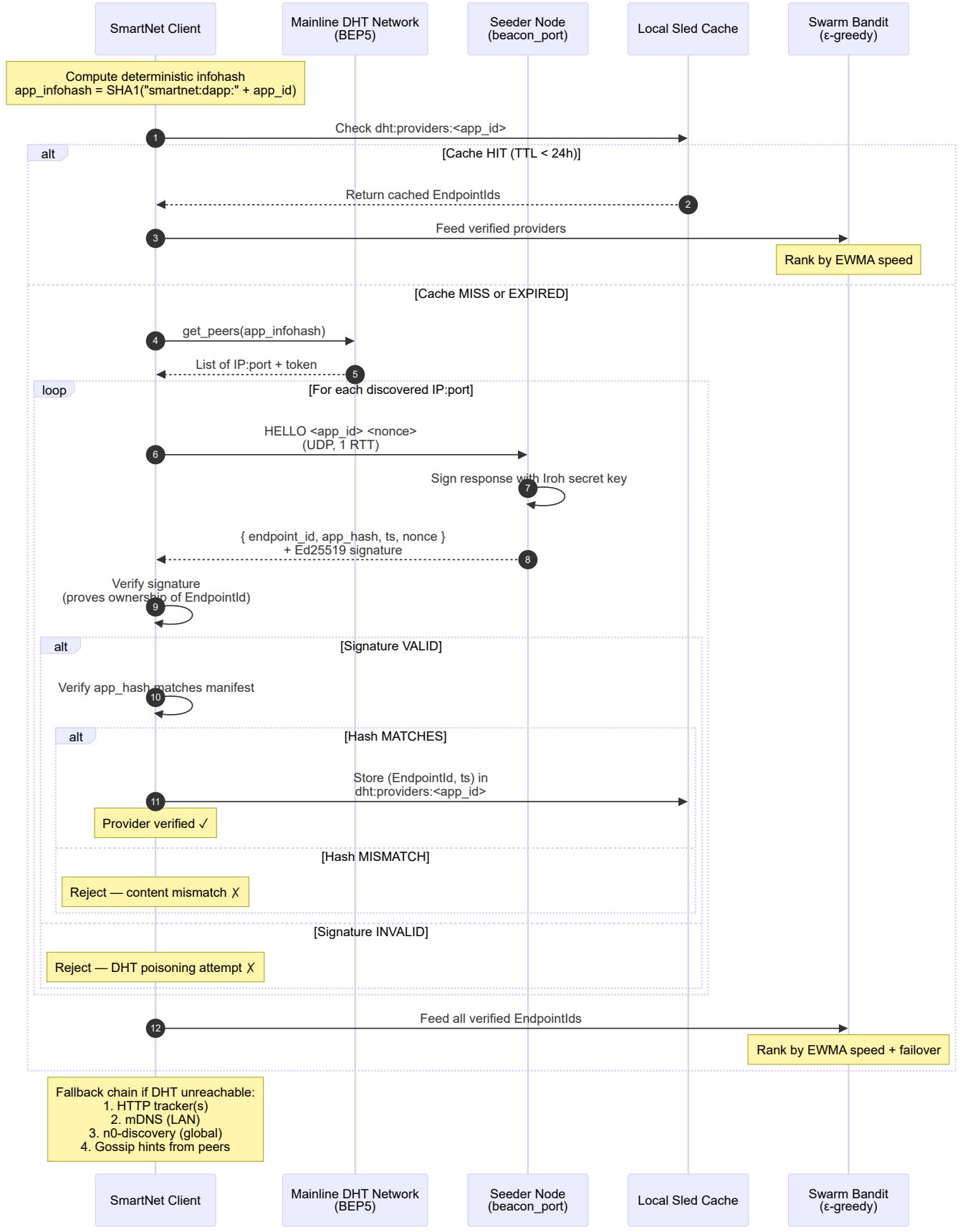
   $$app\_infohash = SHA1(''smartnet : dapp :'' + app\_id)$$

   This 20-byte value is known to every peer holding the application manifest.
2. **Seeder announcement.** Every seeder periodically executes `announce_peer(app_infohash, beacon_port, token)` via the Mainline DHT (reusing the existing `librqbit-dht` session from the torrent layer).
3. **Identity-beacon (UDP, 1 RTT).** A lightweight UDP listener on `beacon_port` answers client probes:
   - Client → `HELLO <app_id> <nonce>`
   - Seeder → `{ endpoint_id, app_hash, ts, nonce }` signed with the seeder's Iroh secret key
4. **Cryptographic verification.** The client verifies the signature. This proves the responding peer actually owns the `EndpointId` — preventing DHT poisoning attacks where a malicious actor announces fake IP:port pairs.
5. **Integration.** Verified `EndpointId`s are merged into the existing `gather_providers(app)` pipeline and

fed to the swarm bandit for ranking.

Sequence Diagram: Trackerless DHT Identity-Beacon Resolution

The following diagram illustrates the complete lifecycle of a trackerless peer discovery request, from the initial `app_infohash` computation to the integration of verified providers into the swarm bandit.



Key security properties demonstrated:

- **Cryptographic proof of identity.** The seeder signs the response with its Iroh secret key. A malicious actor announcing fake IP:port pairs in the DHT cannot forge a valid signature for the corresponding `EndpointId` — preventing DHT poisoning attacks.
- **Content integrity.** The `app_hash` in the response is verified against the expected content hash from the manifest. A seeder cannot claim to hold a different application.
- **Rate limiting.** Beacon responses are rate-limited, and DHT hints are merged with throttling to prevent amplification attacks.
- **Additive design.** If the DHT returns zero providers (or is unreachable), the existing tracker/mDNS/n0-discovery logic continues unchanged. The DHT layer is a performance accelerator, not a dependency.

Persistence. Discovered providers are cached in the local `sled` database under `dht:providers:<app_id>` with a 24-hour TTL. Subsequent installations of the same dApp are near-instant and survive temporary DHT unavailability.

Fallback chain. If DHT is unreachable (e.g., symmetric NAT blocking UDP), the client gracefully falls back to:

1. HTTP tracker(s) → 2. mDNS (LAN) → 3. n0-discovery (global) → 4. Gossip hints from connected peers.

Security properties.

- Signed beacon prevents fake provider injection.
- Rate-limited beacon responses and merged DHT hints prevent amplification.
- `app_hash` in the response is verified against the expected content hash.
- The DHT layer is additive — if it returns zero providers, the existing tracker/mDNS logic continues unchanged.

This design removes the HTTP tracker as a hard dependency for dApp discovery. The tracker becomes a **bootstrap accelerator**, not a gatekeeper. Combined with mDNS and n0-discovery, the network can operate fully trackerless while maintaining sub-second peer resolution.

10.6 Network Map & Byte-Hours Leaderboard

The tracker includes a built-in Network Map system that ranks seeders by their contribution to the network.

Metrics scraping:

1. Seeders expose a Prometheus-compatible `/metrics` endpoint (default port 9466).
2. The tracker's scraper polls each `metricsUrl` every 60 seconds.
3. Metrics are aggregated: `appsSeeded`, `gbSeeded`, `uptimeSecs`.
4. The Byte-Hours score is computed: $Score = GB_seeded \times (uptime_hours)$.
5. Seeders with no announce for >1 hour are removed from the leaderboard.

Dashboard:
The tracker serves an HTML dashboard at `GET /map` with auto-refresh every 5 seconds, displaying the live leaderboard. The same data is exposed as JSON at `GET /seeders` for integration into the desktop client's Network Map tab.

Lock-free metrics architecture:
Inside `smartnet-seeder`, the metrics structure is implemented using Rust's `std::sync::atomic` types. Data collection for logs and external requests occurs without heavy locks (`Mutex` / `RwLock`), guaranteeing zero overhead under high load.

10.6.1 Personal Speed Metrics & 3D Visualization

Beyond the global Byte-Hours leaderboard, the Network Map provides **personalized, real-time feedback** to each user about their own connection quality to every seeder in the network.

"MY SPEED" column. Next to the global Byte-Hours (GB·h) column, the Network Map displays a **"MY SPEED"** column — the user's personally measured download speed from each seeder (matched by `endpointId`). This is rendered as a compact progress bar with success markers (✓), or `–` for seeders the user has never downloaded from. This directly ties the global seeder ranking to the user's real-world experience.

3D Globe coloring by personal speed. The Network Map's 3D globe supports two coloring modes, switchable via a toggle above the legend:

- **BY SOURCE** (default): nodes colored by transport type / discovery source.
- **BY MY SPEED**: nodes colored by the bandit's measured performance for the current user:
  - 🟢 **Bright green** — fast
  - 🔵 **Cyan** — medium
  - 🔵 **Muted blue** — slow
  - 🟠 **Orange** — had errors
  - ⚪ **Gray** — no personal data yet

Hover tooltip. Hovering over any node on the globe reveals a tooltip with: node label, personal speed as a percentage of the user's maximum observed speed, and success/failure counts (or "no data"). The metric is visible directly on the point — no cross-referencing with the table required.

Top Seeders widget. In Settings → P2P, below the swarm toggle, a **"Top Seeders by Speed"** widget displays the highest-performing peers ranked by the bandit's weights, with speed bars and success counters. This gives users immediate visibility into which seeders are delivering the best experience.

System Console logging. After real downloads, the top-3 seeders are logged to the System Console (rate-limited to once per 60 seconds, only when statistics exist):
🏆 top-seeders: 1. ab12...cd34 1234 B/s · 2. ef56...7890 980 B/s · ...

This transforms the Network Map from a passive leaderboard into an **active diagnostic tool** — users can visually identify fast routes, detect problematic peers, and understand their personal position within the symbiotic network.

10.7 Self-Hosted Tracker (B2B Scenario)

Netfory enables a business model around tracker operation. Any entity can deploy a self-hosted tracker, creating a B2B infrastructure layer.

Minimum requirements for a compliant tracker:

1. Accept `POST /seeder/heartbeat` with secp256k1 signature verification and maintain a per-address uptime window.

2. Serve `GET /seeder/heartbeat/<address>` → `{ uptime, k_up }`.
3. Serve `GET /apps` → list of applications with boost-pools.
4. (Optional, roadmap) Flag each discovered dApp as `paid` for store illustration.

Business opportunities:

- **Premium listing.** Charge developers for featured placement in the tracker's recommendations.
- **Availability analytics.** Sell uptime reports and seeder health dashboards to dApp authors.
- **Boost-pool ranking.** Offer premium ranking based on boost-pool size.
- **Private tracker networks.** Operate invite-only trackers for enterprise or community use cases.

This creates a permissionless market for coordination services, where trackers compete on reliability, speed, and value-added features.

10.8 Privacy Guarantees

The tracker is designed to minimize the exposure of user data.

What the tracker sees:

- Public addresses (`S...`) of seeders.
- Content hashes (`app_id`, `hash`, `merkle`) of announced dApps.
- `endpointId` (Iroh network address) for peer discovery.
- `metricsUrl` (optional, for Network Map).

What the tracker does NOT see:

- Message contents (E2EE messages travel directly via Iroh Gossipsub).
- Private data (keys, seeds, personal files).
- User browsing history (which dApps a user downloads).
- IP addresses (seeders connect via Iroh's relay mesh; direct IPs are not exposed to the tracker).

No telemetry.
The tracker does not collect analytics about user behavior. It does not log which clients download which dApps. It only records announcements from seeders.

10.9 Summary

The Tracker layer of Netfory achieves:

- **Coordination without control.** The tracker accelerates discovery but does not store content or act as a trust anchor.
- **Cryptographic integrity.** All announcements are signed; content is verified client-side via BLAKE3. The tracker cannot fake content.
- **Resilience.** Multi-tracker fallback ensures the network survives tracker failures. Pure P2P discovery (mDNS + n0) works without any tracker.
- **Transparency.** Byte-Hours leaderboard provides verifiable proof of seeder contribution.
- **Privacy.** The tracker sees only public metadata — no message contents, no private data, no browsing

history.
- **B2B opportunity.** Self-hosted trackers enable a permissionless market for coordination services.

The tracker is not a server. It is a lightweight coordinator — a telephone directory for the P2P network. It tells you who has what, but it cannot alter what they have. The mathematics of BLAKE3 and secp256k1 ensure that content proves itself, independent of any coordinator.

This is the essence of "Coordination, Not Control": the network organizes itself through cryptographic consensus, and the tracker merely observes and reports.

---

11. Privacy & Security Architecture

11.1 Application Sandboxing (Zero Local Access)

In centralized ecosystems, executing third-party code requires implicit trust in the hosting provider and the application developer. Netfory eliminates this trust requirement through strict application sandboxing.

All decentralized applications (dApps) execute within a strictly limited WebView container managed by the Tauri v2 core. The sandbox enforces two absolute rules:

1. **Zero Local Access.** The dApp has no access to the host machine's filesystem, system processes, or environment variables. It cannot read user files, scan the local network, or execute arbitrary system commands. All persistent data generated by the dApp is confined to an isolated `IndexedDB` instance within the WebView context.
2. **No Raw Sockets.** The dApp cannot open arbitrary network connections (e.g., TCP/UDP sockets) to external servers. All network interactions must route exclusively through the Netfory P2P transport abstraction (`window.smartholdem` API).

Cryptographic operations (signing transactions, decrypting the local vault) are executed in a background thread (`spawn_blocking`) within the main Rust process. The JavaScript context never handles plaintext private keys or mnemonic phrases. Context isolation is mathematically enforced by the browser engine and the Tauri IPC boundary.

11.2 Network Stealth & DDoS Protection

Traditional hosting requires operators to expose public IP addresses and open inbound ports (e.g., 80, 443) to the global internet, making them trivial targets for DDoS attacks and state-level censorship.

Netfory seeders operate in **Network Stealth** mode. Because the transport layer is built on Iroh (QUIC), seeders do not require public IP addresses or port forwarding. Connectivity is established via:

- **NAT Traversal (Hole Punching):** STUN/TURN protocols allow peers behind strict carrier-grade NATs or hardware firewalls to establish direct peer-to-peer connections.
- **DERP Relay Mesh:** If direct hole-punching fails (e.g., symmetric NAT), traffic is routed through a decentralized mesh of Designated Encrypted Relay for Packets (DERP) nodes.

The relay is not a centralized server; any peer can operate a relay. Crucially, the seeder's real IP address is never exposed to the downloading client or the public internet. The network sees only the seeder's cryptographic `endpointId`.

11.3 Content Poisoning Protection

A critical barrier to decentralized hosting adoption is the operator's fear of unknowingly hosting illegal or malicious content. Netfory solves this through cryptographic obfuscation and community-driven consensus.

1. Fragmented, Encrypted Blobs.
Data on a seeder's disk is not stored as recognizable files (e.g., `.mp4`, `.exe`). It is stored as fragmented, encrypted chunks (blobs) addressed solely by their BLAKE3 cryptographic hash. The seeder operator physically cannot inspect, read, or modify the content they are redistributing. They are merely holding cryptographically sealed shards.

2. BLAKE3 Hash Tree Verification.
Every chunk is independently verified against a BLAKE3 hash tree. If a malicious actor attempts to inject corrupted data into the swarm, the hash verification fails instantly, and the client rejects the chunk. Content proves itself; it does not rely on the honesty of the provider.

3. DPoS Blacklist & Auto-Eviction.
If the community identifies a specific `app_id` as malicious (e.g., malware, illegal content), a decentralized vote is initiated on the SmartHoldem DPoS blockchain. Once the vote passes:

- Trackers cease announcing the `app_id` in their `/apps` catalog.
- Seeders automatically detect the blacklist status and purge the associated blobs from their local storage via the garbage collection (GC) eviction policy.

The seeder is protected by mathematics: they cannot be held liable for content they cannot read, and the network provides an automated mechanism to purge flagged data.

11.4 End-to-End Encryption (E2EE) & Local Vault Security

All user communications and local data are secured by military-grade cryptography, ensuring that privacy is a default property of the protocol, not an optional feature.

Messaging Encryption:

- **Key Exchange:** X25519 (Elliptic Curve Diffie-Hellman) is used to establish a shared secret between peers.
- **Packet Encryption:** Payloads are encrypted using AES-256-GCM. For group chats, the room key is derived from a shared group secret distributed via cryptographically signed invite links.
- **Transit:** Messages travel directly via Iroh Gossipsub. No central server stores, logs, or relays message contents or metadata.

Local Vault Security:

- The user's 12-word BIP-39 mnemonic and all derived private keys are encrypted locally using AES-256-GCM, protected by a user-defined PIN.
- The plaintext seed never touches the disk.
- **Annihilation:** The protocol includes a `delete_vault` and `wipe_all_chat` function. When invoked, it securely and irreversibly overwrites the local `sled` database, erasing all chat history, P2P subscriptions, and cached keys. This guarantees complete data isolation when switching identities or decommissioning a device.

11.5 Privacy Boundaries of the Tracker

The Tracker is often misunderstood as a surveillance tool. In Netfory, the Tracker's visibility is strictly limited by design.

What the Tracker SEES:

- Public SmartHoldem addresses (`S...`) of announcing seeders.
- Content hashes (`app_id`, `merkle`, `hash`) of announced dApps.
- Iroh `endpointId` for peer discovery.
- Optional `metricsUrl` for the Network Map leaderboard.

What the Tracker DOES NOT SEE:

- Message contents or chat metadata (E2EE messages bypass the tracker entirely).
- Private user data, files, or browsing history (which dApps a user downloads).
- Real IP addresses of seeders (due to Iroh QUIC hole-punching and relay abstraction).

The Tracker is a telephone directory, not a wiretap. It facilitates discovery but possesses neither the capability nor the cryptographic keys to intercept or alter the data flowing through the network.

11.6 Summary

The Privacy & Security Architecture of Netfory achieves:

- **Absolute Application Isolation.** Zero Local Access and No Raw Sockets prevent dApps from compromising the host machine.
- **Infrastructure Stealth.** Seeders operate without exposing public IPs, neutralizing DDoS and censorship vectors.
- **Operator Immunity.** Fragmented, BLAKE3-verified blobs and DPoS blacklists ensure seeders cannot be forced to host or inspect malicious content.
- **Mathematical Privacy.** X25519 + AES-256-GCM E2EE, local AES-GCM vault encryption, and secure annihilation guarantee that user data remains exclusively under user control.
- **Minimal Tracker Exposure.** The coordination layer sees only public metadata, preserving the anonymity and privacy of network participants.

In Netfory, security is not a policy enforced by a corporation. It is a structural property enforced by cryptography, sandboxing, and decentralized consensus.

---

12. LAN-First Autonomy

12.1 The Fragility of Global Dependency

The resilience of a decentralized network is measured not by its performance under ideal conditions, but by its survival during catastrophic fragmentation. Traditional architectures exhibit a critical vulnerability: they assume perpetual, unrestricted access to the global internet.

- **Clearnet:** Relies on centralized DNS resolvers and BGP routing. A single cable cut, BGP hijack, or state-

level firewall (e.g., TSPU) instantly severs access.

- **Tor / I2P:** Depend on global directory authorities and introducer nodes to bootstrap circuits. Without access to these external directories, the network cannot form.
- **IPFS:** While theoretically peer-to-peer, practical usage relies heavily on centralized HTTP gateways or globally reachable bootstrap nodes. In an isolated network, content resolution fails.

When the global internet is partitioned, these networks cease to function. Netfory is explicitly architected to reject this fragility.

12.2 The LAN-First Paradigm

Netfory inverts the traditional dependency model. Global internet connectivity is treated as an optional enhancement for broader discovery, not a fundamental requirement for operation.

The network is designed to function optimally within a Local Area Network (LAN). If a city, region, or facility is disconnected from the global internet, Netfory does not degrade into a non-functional state. Instead, it gracefully transitions into a fully autonomous, self-sustaining local mesh.

12.3 Technical Mechanism: mDNS and Local Discovery

When global connectivity is lost, the Netfory client automatically falls back to **Multicast DNS (mDNS)**, operating on UDP port 5353 within the local subnet (e.g., `192.168.x.x`).

Discovery Process:

1. Each active Netfory node periodically broadcasts a multicast packet containing its Iroh `endpointId` and a list of locally cached `app_id` hashes.
2. Other nodes on the same LAN listen for these broadcasts and update their local peer tables.
3. Direct, encrypted QUIC connections are established between local peers without traversing any Wide Area Network (WAN) infrastructure.
4. No port forwarding, public IP addresses, or external relays are required.

This mechanism ensures that peer discovery and content routing continue seamlessly, bounded only by the physical limits of the local network switch or Wi-Fi router.

12.4 Local State Persistence

Autonomy requires that the client does not depend on external servers to maintain user state. Netfory achieves this through the local `sled` embedded key-value database.

All critical operational data is persisted locally on the user's device:

- **Identity Vault:** Encrypted BIP-39 seed and derived keys.
- **E2EE Chat Logs:** Complete message history and group chat states.
- **P2P Subscriptions:** Lists of followed users and trusted contacts.
- **Cached dApps:** Locally installed and verified application blobs.

Because this state is local, the user experience remains uninterrupted during an internet blackout. When global connectivity is eventually restored, the client automatically synchronizes pending transactions, fetches new global peer lists via n0-discovery, and updates the local `sled` database with the latest network state.

12.5 Operational Scenarios in Isolation

To demonstrate the practical impact of LAN-first autonomy, consider three concrete scenarios where global connectivity is severed.

Scenario 1: Regional Infrastructure Failure
A natural disaster or accidental fiber cut severs a city's connection to the global internet backbone.

- *Clearnet/Tor/I2P:* Instantly dead. Communication halts.
- *Netfory:* Nodes within the city's local ISPs or community Wi-Fi networks automatically discover each other via mDNS. Residents continue to chat via E2EE, share emergency files via CID, and run locally cached dApps without interruption.

Scenario 2: State-Level Deep Packet Inspection (TSPU)
A government activates aggressive traffic shaping to block all unrecognized P2P protocols and foreign IP addresses.

- *Clearnet:* Blocked.
- *Netfory:* Because local mDNS traffic never leaves the `192.168.x.x` subnet, it is invisible to upstream DPI systems. The local mesh operates normally, shielded by the physical boundary of the local router.

Scenario 3: Corporate or Campus Blackout
An enterprise or university completely disables external internet access for security or policy reasons.

- *Netfory:* Employees or students on the internal network can still use the Netfory client to share large files, collaborate on DevHub repositories (if locally seeded), and communicate securely, utilizing the existing local infrastructure without requiring IT to open external firewall ports.

12.6 Summary

The LAN-First Autonomy architecture of Netfory achieves:

- **Graceful Degradation.** The network seamlessly transitions from global P2P to local mesh without user intervention or loss of functionality.
- **Zero External Dependency.** Operation does not rely on global DNS, directory authorities, or centralized bootstrap nodes.
- **Cryptographic Continuity.** Local `sled` persistence ensures that identity, chat history, and application state remain fully accessible and functional during isolation.
- **Censorship Resistance.** Local mDNS traffic is inherently invisible to upstream Deep Packet Inspection (DPI) and national firewalls.

Netfory does not merely survive network partitioning; it is designed to thrive within it. The network is not a destination you connect to; it is an environment you inhabit, regardless of the state of the global internet.

---

13. Web 4.0: The Symbiotic Paradigm

13.1 The Three Paradigms of the Web

The evolution of the internet can be understood through three distinct paradigms, each defined by the relationship between the user and the network.

Web 2.0 — The Platform Paradigm.
The user is a consumer. Data resides on corporate servers. Identity is granted by platforms. Value flows to intermediaries. The network is a place you visit.

Web 3.0 — The Ownership Paradigm.
The user is an owner. Tokens and smart contracts reside on blockchains. Yet the frontend remains centralized. The user owns assets but not the access layer. The network is a place you visit, but you hold the keys to the door.

Web 4.0 — The Symbiotic Paradigm.
The user is the infrastructure. Every client is an autonomous node. Every device is a peer. Every action is a protocol interaction, not an API call. The network is not a place you visit; it is an environment you inhabit. You do not use the network — you **are** the network.

The distinction is not semantic. It is architectural. Web 3.0 decentralizes the ledger. Web 4.0 decentralizes the entire stack — transport, identity, execution, economy, and discovery — into a single, self-sustaining, agent-native infrastructure.

13.2 The Four Formulas of Web 4.0

Netfory is built upon four foundational formulas that define the Web 4.0 paradigm. These are not marketing slogans; they are architectural invariants enforced by the protocol.

Formula 1: Protocol Stack Replaces API

In Web 2.0 and Web 3.0, applications communicate with servers via HTTP/REST APIs. Access is a matter of **permission** — the server decides whether to respond.

In Web 4.0, applications communicate via protocol stacks (`sn://`, `u://`, `sth://`, `dev://`). Access is a matter of **connection** — the protocol decides how to route the request.

Consequence: No server can deny access. No API key can be revoked. No rate limit can be imposed by a central authority. The protocol is the law.

Formula 2: State-First Distribution

In Web 2.0 and Web 3.0, the user "downloads" a page or "fetches" data from a server. State exists on the server; the client is a thin viewer.

In Web 4.0, state exists distributed in the network. The user does not download a page; they **synchronize state** with the network via P2P. The client is a thick participant.

Consequence: The network does not have a "server" to query. It has peers to synchronize with. If one peer goes offline, the state is still accessible from others. There is no single source of truth — the truth is distributed.

Formula 3: Zero-Infra

In Web 2.0 and Web 3.0, applications require infrastructure: AWS, Google Cloud, Vercel, Cloudflare. The backend logic (databases, APIs, load balancers) lives on centralized servers.

In Web 4.0, all backend logic migrates to the P2P protocol layer. There is no AWS. There is no Vercel. There is no Cloudflare. The network itself is the infrastructure.

Consequence: No corporation can deplatform an application. No government can seize a server. No provider can impose terms of service. The infrastructure is the users.

Formula 4: Content-Addressability Everywhere

In Web 2.0 and Web 3.0, content is addressed by location (URL, domain, IP address). A domain can be seized. A certificate can be revoked. A server can be deplatformed.

In Web 4.0, everything is bound to a Content Identifier (CID), computed via BLAKE3. Content proves itself. No domain can be seized. No certificate can be revoked. No server can be deplatformed.

Consequence: Content is immortal. As long as one peer in the world holds the blob, the content exists. The network is censorship-resistant by mathematical design, not by policy.

13.3 The Symbiotic Network

The term "symbiotic" is not metaphorical. It describes the precise relationship between the user and the network.

In Web 2.0, the relationship is **parasitic**: the platform extracts value from the user (data, attention, money) and returns a service. The user is the product.

In Web 3.0, the relationship is **transactional**: the user owns tokens and interacts with smart contracts. The platform extracts fees (gas, commissions). The user is a customer.

In Web 4.0, the relationship is **symbiotic**: the user contributes to the network (storage, bandwidth, computation) and receives value (access, rewards, sovereignty). The user is a participant.

The Symbiosis Loop:

1. The user runs a Netfory client.
2. The client discovers peers, verifies content, and redistributes blobs.
3. The client earns STH for providing storage and bandwidth.
4. The user spends STH on names, dApps, and services.
5. The network grows stronger, more resilient, more valuable.
6. The user benefits from a more powerful network.

This is not a one-way extraction. It is a feedback loop. The user and the network evolve together. The network is not a tool the user wields; it is an organism the user inhabits.

13.4 The User as Infrastructure

In Web 2.0 and Web 3.0, the user is separate from the infrastructure. The user consumes; the infrastructure serves.

In Web 4.0, the user **is** the infrastructure. Every client is a node. Every device is a server. Every user is a seeder.

Implications:

- **No central point of failure.** There is no server to seize. There is no data center to raid. There is no CEO to subpoena. The infrastructure is distributed across millions of independent devices.
- **No single point of control.** No corporation can dictate terms. No government can impose censorship. No regulator can enforce compliance. The infrastructure is governed by protocol, not by policy.
- **No single point of extraction.** No platform can charge a 30% fee. No intermediary can impose a tax. No middleman can skim value. The infrastructure is funded by the users, for the users.

The user is not a consumer. The user is not a customer. The user is the network.

13.5 Economic Consequences

The Web 4.0 paradigm fundamentally alters the economics of the internet.

The Death of the Platform Tax.
In Web 2.0 and Web 3.0, platforms extract value: 30% app store fees, 3% payment processor fees, gas fees, commission fees. The platform is the tollbooth.

In Web 4.0, there is no platform. There is only the protocol. Money flows directly from user to creator. There is no tollbooth. There is no middleman. There is no tax.

The Rise of the Micro-Economy.
In Web 2.0 and Web 3.0, only large-scale operations are profitable. The platform tax makes small-scale monetization unviable.

In Web 4.0, the zero-tax economy enables micro-monetization. A user can earn STH for seeding a single dApp. A developer can earn STH for publishing a single application. A creator can earn STH for a single subscription. The economy is democratized.

The Alignment of Incentives.
In Web 2.0 and Web 3.0, the platform's incentives are misaligned with the user's. The platform profits from extraction; the user profits from usage.

In Web 4.0, the incentives are aligned. The user profits from contributing (seeding, creating, participating). The network profits from the user's contribution. The user and the network are symbiotic.

13.6 The End of Intermediaries

The Web 4.0 paradigm renders intermediaries obsolete.

No Hosting Provider.
Applications are distributed via P2P. There is no server to host. There is no CDN to cache. There is no cloud to store.

No Identity Provider.
Identity is self-sovereign, anchored on the blockchain. There is no OAuth. There is no email verification. There is no phone number.

No Payment Processor.
Payments are peer-to-peer, on-chain. There is no Stripe. There is no PayPal. There is no bank.

No Content Moderator.
Content is verified by cryptography, not by policy. There is no moderation team. There is no terms of service. There is no community guidelines.

No Domain Registrar.
Domains are registered on-chain, first-win. There is no ICANN. There is no GoDaddy. There is no Namecheap.

The intermediary is the single point of failure. The intermediary is the single point of control. The intermediary is the single point of extraction. Web 4.0 eliminates the intermediary.

13.7 Summary

The Web 4.0 paradigm of Netfory achieves:

- **Protocol over permission.** Access is a matter of connection, not authorization.
- **State over storage.** The user synchronizes with the network, not downloads from a server.
- **Zero-infra over centralized hosting.** The network is the infrastructure.
- **Content-addressability over location.** Content proves itself; it cannot be seized or revoked.
- **Symbiosis over extraction.** The user and the network evolve together.
- **The user as infrastructure.** Every client is a node. Every device is a server. Every user is the network.
- **The end of intermediaries.** No hosting provider, no identity provider, no payment processor, no content moderator, no domain registrar.

Web 4.0 is not an evolution of Web 3.0. It is a paradigm shift. Web 3.0 decentralizes the ledger. Web 4.0 decentralizes the entire stack. Web 3.0 gives the user ownership. Web 4.0 gives the user sovereignty. Web 3.0 is a place you visit. Web 4.0 is an environment you inhabit.

Netfory is the first implementation of Web 4.0. This is not a concept. It is not a roadmap. It is a working, tested, production-ready stack. The symbiotic web is live.

---

14. Conclusion

The evolution of the internet has been constrained by a persistent architectural vulnerability: the centralization of the access layer. While Web 3.0 successfully decentralized the ledger, it left the frontend exposed to corporate infrastructure, regulatory seizure, and single points of failure. Existing peer-to-peer alternatives failed to close the loop, lacking either the performance, the economic incentives, or the human-readable addressing required for mass adoption.

Netfory resolves this crisis by introducing a fully autonomous, agent-native Web 4.0 infrastructure. It replaces the centralized hosting stack with a decentralized, content-addressed distribution network. By unifying the Iroh (QUIC) transport layer, BLAKE3 cryptographic verification, Self-Decentralized Identity (SDI), and a native Proof-of-Utility economic model, Netfory creates a self-sustaining ecosystem. Money flows directly from users to creators of value — authors, developers, and seeders — without platform extraction or intermediary taxation.

This architecture represents a fundamental paradigm shift. In Netfory, the network is not managed by centralized agents; the network **is** the agents. Every user's client is an autonomous node that discovers peers, verifies content, negotiates routes, and sustains the economy. This is realized through four foundational invariants: the protocol stack replaces the API, state is distributed first, infrastructure is zero, and content-addressability is absolute.

Crucially, Netfory is not a theoretical roadmap, a conceptual whitepaper awaiting implementation, or a speculative token launch. It is a working, tested, production-ready stack. The Tauri v2 desktop client, the headless Rust seeder, the lightweight `sled`-based tracker, the `git-remote-dev` helper, and the SmartHoldem DPoS blockchain are fully integrated and operational. The mathematics of BLAKE3, secp256k1, and QUIC are not proposals; they are the active, verifiable mechanics of the network.

The era of renting access to the internet is ending. Netfory provides the infrastructure for an internet that is owned, operated, and sustained by its users. We have built the transport layer of the symbiotic web.

The network is live.

---

15. References

The architecture of Netfory is built upon established cryptographic primitives, peer-reviewed network protocols, and robust systems engineering standards. The following references form the foundational literature and technical specifications for the Web 4.0 protocol stack.

15.1 Cryptographic Standards & Primitives

1. **Nakamoto, S. (2008).** *Bitcoin: A Peer-to-Peer Electronic Cash System.* (Foundational paradigm for decentralized trust and cryptographic ownership).
2. **Aumasson, J.-P., et al. (2023).** *BLAKE3: A Fast, Secure, and Simple Hash Function.* (Provides the 25× speed advantage over SHA-256 and real-time content integrity verification).
3. **Bernstein, D. J. (2006).** *Curve25519: New Diffie-Hellman Speed Records.* (Basis for X25519 Elliptic Curve Diffie-Hellman key exchange used in E2EE messaging).
4. **Dworkin, M. (2007).** *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* NIST Special Publication 800-38D. (Standard for AES-256-GCM local vault and message encryption).
5. **Dobbertin, H., Bosselaers, A., & Preneel, B. (1996).** *RIPEMD-160: A Strengthened Version of RIPEMD.* (Used in conjunction with SHA-256 for SmartHoldem address derivation).
6. **National Institute of Standards and Technology (NIST). (2015).** *Secure Hash Standard (SHS).* FIPS PUB 180-4. (SHA-256 baseline for Merkle tree construction in storage proofs).

15.2 Wallet & Identity Standards (BIPs)

7. **Palatinus, M., & Rusnak, P. (2012).** *BIP-0039: Mnemonic code for generating deterministic keys.* (12-word seed phrase generation).
8. **Wuille, P. (2012).** *BIP-0032: Hierarchical Deterministic Wallets.* (Isolated key derivation for distinct subsystems).
9. **Davies, A., et al. (2014).** *BIP-0044: Multi-Account Hierarchy for Deterministic Wallets.* (Standardized path derivation `m/44'/255'/0'/1/i` for dApp isolation).

15.3 Network Protocols & P2P Architecture

10. **Iroh Development Team (n0).** *Iroh: A peer-to-peer networking stack built on QUIC.* (Core transport layer replacing legacy libp2p/IPFS, enabling NAT traversal and async multiplexing).

11. **Iyengar, J., & Thomson, M. (2021).** *QUIC: A UDP-Based Multiplexed and Secure Transport.* RFC 9000. (Foundation for 0-RTT connection resumption and head-of-line blocking elimination).
12. **Rosenberg, J., et al. (2003).** *STUN: Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs).* RFC 3489. (Enables direct P2P hole-punching for Network Stealth).
13. **Cheshire, S., et al. (2013).** *Multicast DNS.* RFC 6762. (Enables LAN-First Autonomy and local peer discovery without global internet).
14. **Maymounkov, P., & Mazieres, D. (2002).** *Kademlia: A Peer-to-peer Information System Based on the XOR Metric.* (Conceptual basis for Mainline DHT branch pointers in DevHub).

15.4 Blockchain & Economic Consensus

15. **SmartHoldem Core Team. (2026).** *SmartHoldem DPoS Protocol Specification.* (Details on block generation, delegate validation of `StorageProofTx`, and low-fee transaction mechanics).
16. **SmartHoldem Core Team. (2026).** *Transaction VendorField Specification.* (Mechanism for anchoring metadata, such as `u://` names and `sth_boost:<app_id>`, directly on-chain).
17. **Buterin, V. (2014).** *A Next-Generation Smart Contract and Decentralized Application Platform.* (Contextual contrast: highlights the Web 3.0 frontend centralization problem that Netfory resolves).

15.5 Systems Engineering & Development Frameworks

18. **Matsakis, N. D., & Klock, F. S. (2014).** *The Rust Programming Language.* (Core language for Tauri v2, headless seeder, and tracker, ensuring memory safety and zero-cost abstractions).
19. **Stjepanović, A. (2020).** *sled: A modern, high-performance embedded key-value database in Rust.* (Persistent storage engine for the Tracker and local client state).
20. **Torvalds, L. (2005).** *Git: A Fast Version Control System.* (Foundation for the `git-remote-dev` helper and decentralized source code distribution).
21. **W3C Web Application Security Working Group. (2021).** *Content Security Policy Level 3.* (Conceptual basis for WebView sandboxing and Context Isolation in Tauri).

16. Roadmap & Next Milestones

Netfory is not a theoretical roadmap. It is a working, tested, production-ready stack. However, the evolution of the symbiotic web is continuous. The following phases outline the strategic trajectory of the protocol, grounded in shipped code and measurable deliverables.

16.1 Phase 1 — Foundation (Shipped, 2026)

The core infrastructure is live and operational. All components listed below are deployed, tested, and actively used by the network:

- **Core P2P Stack.** Iroh (QUIC) transport layer with STUN/TURN/DERP hole-punching, mDNS for LAN-first autonomy, and `n1-relay` decentralized relay mesh.
- **Content Addressing.** BLAKE3 hash trees for immutable, censorship-resistant blob distribution.
- **Identity & Naming.** Self-Decentralized Identity (SDI) with BIP-39/BIP-32/BIP-44 key derivation, on-chain `u://` and `.sth` registration, procedural avatars.
- **SmartNet App SDK.** Native `window.smartholdem` bridge with `useSmartNetWallet()`, `usePayments()`,

and `useStorage()` hooks for dApp developers.

- **DevHub & Git Remote Helper.** `git-remote-dev` utility enabling transparent `dev://` cloning and pushing via Mainline DHT, with IDE integration (VS Code, WebStorm, GitKraken).
- **Swarm Loading (Multi-Armed Bandit).** ε-greedy algorithm with EWMA (α=0.2) for intelligent seeder selection, persistent weights in `sled`, and 3D Network Map visualization with "MY SPEED" column.
- **Trackerless Discovery.** Mainline DHT (BEP5) integration with signed identity-beacon for trackerless peer resolution.
- **Proof-of-Utility Economics.** Signed heartbeats, 24-hour sliding window (288 slots), K_{up} uptime coefficient, Global Seeding Pool, and Boost-Pool market mechanism.
- **Proof-of-Storage.** Merkle spot-checking with blockchain entropy and unique sealing (PoRep) for Sybil resistance.
- **E2EE Messaging.** X25519 + AES-256-GCM encryption, paid inbox anti-spam, self-healing group chats via `sn://group/` links.
- **Neural Translation Layer.** Local INT8/INT4 quantized model for offline, privacy-preserving cross-lingual content rendering.

16.2 Phase 2 — Expansion (Q4 2026 — Q1 2027)

The next phase focuses on deepening the economic loop, improving distribution efficiency, and expanding the marketplace:

- **Full HashSeq Per-File Distribution.** Migration from single-blob dApp archives to iroh collections / HashSeq, enabling chunk-level parallelism, deduplication across applications, and partial updates.
- **Secondary Marketplace for `.sth` Domains.** On-chain auctions and peer-to-peer trading of registered domain names, with royalty fees burned on each transaction.
- **On-Chain Verifiable Receipts.** Cryptographic proof-of-payment for paid group entry, subscriptions, and in-app purchases, enabling trustless dispute resolution.
- **Availability Staking.** Seeders stake STH as collateral for uptime guarantees; slashing mechanism for downtime creates financial accountability.
- **Proof-of-Storage Slashing.** Staked collateral destroyed for invalid storage proofs, reinforcing the economic integrity of the storage layer.
- **I2P / Tor Routing Integration.** Optional hidden-network routing for absolute metadata privacy, enabling seeders to operate without exposing any network-level fingerprint.
- **Mobile Client (Alpha).** Tauri v2 port to iOS and Android, extending the symbiotic web to mobile devices with native P2P transport.

16.3 Phase 3 — Sovereignty (2027+)

The long-term vision focuses on cross-chain interoperability, decentralized reputation, and the evolution of the network into a global infrastructure layer:

- **Decentralized Identity Reputation Scoring.** On-chain reputation derived from seeder uptime, dApp quality, and community contributions — enabling trustless peer selection without centralized rating agencies.
- **Cross-Chain Bridges.** Native bridges to BSC, Ethereum, and Solana, enabling STH to flow across

ecosystems while preserving the SmartHoldem DPoS as the trust anchor for Netfory.

- **Decentralized Data Center (DePIN).** As the network scales, everyday users can monetize their idle storage and bandwidth by renting it directly to corporations and enterprises. Netfory transforms into a global, censorship-resistant alternative to traditional cloud providers (AWS/Cloudflare), with all B2B resource settlements executed in $STH.
- **Advanced Traffic Anonymization.** Research and integration of routing support through hidden networks (I2P / Tor layers) to ensure absolute network confidentiality for nodes, building upon the existing decentralized DERP relay mesh.
- **Governance Layer.** On-chain voting for protocol upgrades, blacklist decisions, and economic parameter adjustments — executed by STH holders with reputation-weighted voting power.
- **Local-First AI Primitives.** Expansion of the existing native neural layer (currently INT8/INT4 translation) into additional on-device AI capabilities: content classification, spam detection, smart routing heuristics, and autonomous seeder reputation scoring — all running locally without cloud dependencies, preserving the "zero-trust, zero-cloud" architecture.

16.4 The Unchanging Core

Regardless of the phase, the following architectural invariants remain permanent:

1. **The network IS the agents.** No server-based manager agents. No cloud-hosted orchestrators. Every user's client is an autonomous node.
2. **Content proves itself.** Everything is bound to a BLAKE3 CID. No domain can be seized. No certificate can be revoked.
3. **Zero infrastructure.** No AWS. No Google Cloud. No Vercel. The network itself is the infrastructure.
4. **Protocol over permission.** Access is a matter of connection (`sn://`, `u://`, `sth://`, `dev://`), not authorization.
5. **Deflationary by design.** STH is the physical fuel of a decentralized infrastructure, not a speculative asset.

The era of renting access to the internet is ending. Netfory provides the infrastructure for an internet that is owned, operated, and sustained by its users.

The network is live. The roadmap is being built.

Appendix A: Glossary of Terms

This glossary provides concise definitions of the technical terms, protocols, and architectural components used throughout this White Paper. Terms are listed alphabetically.

AES-256-GCM. A symmetric encryption algorithm (Advanced Encryption Standard, 256-bit key, Galois/Counter Mode) used in Netfory for end-to-end encrypted messaging. Provides both confidentiality and authenticity in a single pass.

Availability Staking. A mechanism (roadmap) whereby seeders lock STH as collateral to guarantee uptime. Downtime or invalid heartbeats result in partial or full slashing of the staked amount.

BIP-39 / BIP-32 / BIP-44. Bitcoin Improvement Proposals defining mnemonic seed phrases (BIP-39), hierarchical deterministic key derivation (BIP-32), and multi-account path structure (BIP-44). Netfory uses these standards for Self-Decentralized Identity (SDI) key management.

BLAKE3. A cryptographic hash function (2020) producing 256-bit digests. Used throughout Netfory for content-addressed blob verification, dApp integrity checks, and Merkle tree construction. Significantly faster than SHA-256 while maintaining equivalent security.

Boost-Pool. A per-application STH pool (`sth_boost:<app_id>`) where developers or users stake tokens to incentivize seeders to prioritize distribution of a specific dApp. Higher boost → higher `priority_weight` in tracker recommendations.

Byte-Hours. The unit of measurement for seeder contribution: gigabytes stored × hours of continuous availability. Tracked via the 288-slot sliding window and visualized in the Network Map leaderboard.

CORS (Cross-Origin Resource Sharing). A browser security mechanism that restricts web pages from making requests to a different origin. SmartNet dApps (origin `sth://<app_id>/`) require backends to return compatible CORS headers.

Content-Addressed. A storage paradigm where data is identified and retrieved by its cryptographic hash (BLAKE3 CID) rather than by location (URL or IP). Guarantees immutability: any modification produces a different address.

Content Identifier (CID). A self-describing, content-addressed hash that uniquely identifies a blob in the Netfory network. Two identical files always produce the same CID; any modification produces a different one.

DHT (Distributed Hash Table). A decentralized key-value lookup system spread across all participating nodes. Netfory uses the BitTorrent Mainline DHT for trackerless peer discovery, eliminating the HTTP tracker as a single point of failure.

DPoS (Delegated Proof of Stake). The consensus mechanism of the SmartHoldem blockchain, on which STH transactions are settled. Provides fast, low-fee finality with secp256k1 (legacy bip-schnorr) signatures.

DERP (Designated Encrypted Relay for Packets). A relay protocol for forwarding encrypted traffic between peers that cannot establish a direct connection (e.g., behind symmetric NAT). Implemented in Netfory as the decentralized `n1-relay` mesh.

Ed25519. An elliptic-curve digital signature algorithm used by Iroh for node identity and by `netfory-provider` for signing API responses. Provides fast verification and small signature sizes.

EndpointId. The 32-byte public key identity of an Iroh node. Unlike IP addresses, EndpointIds are cryptographically stable and do not change when a node's network location changes.

EWMA (Exponentially Weighted Moving Average). A statistical smoothing technique used in the Multi-Armed Bandit algorithm (α = 0.2) to track seeder download speed. Recent measurements receive higher weight while preserving historical context.

Global Seeding Pool. The on-chain STH pool (`SappsPqDQXCRvoi1V2Hs9xycQeT1ptJWV1`) that accumulates dApp publication fees and distributes rewards to active seeders based on their Proof-of-Utility scores.

HashSeq. An iroh collection format for per-file chunked distribution. Intentionally deferred in the current Netfory architecture; the built-in torrent engine handles chunk-level parallelism at the blob level.

Headless Seeder. A minimal, GUI-less Netfory node (`smartnet-seeder`) that runs on VPS or dedicated hardware, providing 24/7 storage and bandwidth without user interaction. Compensated via the Global Seeding Pool.

Identity-Beacon. A lightweight UDP listener (1 RTT) that bridges the gap between DHT-discovered `IP:port` pairs and Iroh `EndpointId`s. Responses are signed with the seeder's Iroh secret key to prevent DHT poisoning.

Iroh. A peer-to-peer networking library (Rust) built on QUIC. Provides encrypted, NAT-traversing connections identified by cryptographic EndpointIds rather than IP addresses. The foundational transport layer of Netfory.

K_up (Uptime Coefficient). A non-linear multiplier (range: 0 to 2.5) applied to seeder rewards based on their 24-hour uptime percentage. 100% uptime → ×2.5 (MAX_K_UP); lower uptime scales linearly downward.

Loopback HTTP. A local-only HTTP channel (127.0.0.1) used for inter-process communication between `git-remote-dev` and the SmartNet client. Authenticated via a random token in `devhub-ipc.json`.

mDNS (Multicast DNS). A zero-configuration LAN discovery protocol. Used by Netfory as a fallback peer discovery mechanism when DHT and HTTP trackers are unreachable.

Monotonic Sequence (seq+1). A strictly incrementing counter signed into every DHT branch pointer during `git push` operations. Prevents repository rollback attacks — the network rejects any publication with a lower sequence number.

Multi-Armed Bandit (ε-greedy). An algorithm for balancing exploration (probing unknown peers) and exploitation (using known-fast peers) during swarm downloads. Default ε = 0.15 (15% exploration probability).

n0-discovery. A global peer discovery service operated by the Iroh project. Used as a fallback in Netfory's discovery chain when DHT, tracker, and mDNS are all unavailable.

n1-relay. A lightweight Rust service implementing the decentralized DERP relay mesh. Forwards encrypted QUIC packets between peers that cannot establish direct connections. Never sees plaintext traffic.

netfory-provider. A P2P-to-clearnet API gateway that translates `api://` requests into standard HTTPS calls and signs responses with an Ed25519 key. Enables dApps to interact with external services while preserving the zero-trust model.

Network Map. A 3D globe visualization in the SmartNet client showing all known seeders, their geographic distribution, transport types, and the user's personal download speed to each. Includes the "MY SPEED" column and Byte-Hours leaderboard.

NodeID. The 64-character hexadecimal representation of an Iroh node's public key. Used as the host component in WebSocket URLs (`ws://<64-hex-nodeId>/...`) and `api://` requests.

PNA (Private Network Access). A browser security policy requiring explicit server consent (`Access-Control-Allow-Private-Network: true`) before allowing requests from public origins to private IP addresses.

PoRep (Proof of Replication). A storage verification mechanism where seeders prove they hold a unique, sealed copy of data (not a symlink or deduplicated reference). Implemented via Merkle spot-checking with blockchain-derived entropy.

PoU (Proof of Utility). The economic consensus mechanism of Netfory: seeders earn STH proportional to their measured contribution (Byte-Hours × K_up), verified via signed heartbeats and the 288-slot sliding window.

Proof-of-Storage. A cryptographic verification layer where the tracker periodically challenges seeders to produce Merkle proofs for random byte ranges of stored blobs. Invalid proofs result in slashing (roadmap).

QUIC. A transport protocol (RFC 9000) providing encrypted, multiplexed, connection-oriented communication over UDP. The foundation of the Iroh transport layer, offering lower latency than TCP+TLS.

Rescue Fallback. A client-side safety mechanism (≥ v1.55.35) that recovers a provider's NodeID from the internal provider map when a dApp uses a malformed WebSocket URL. A safety net, not a contract.

SDI (Self-Decentralized Identity). Netfory's identity system where users generate and control their own cryptographic keypairs (BIP-39 mnemonic → BIP-32 tree → BIP-44 paths). No central authority issues or revokes identities.

secp256k1. The elliptic curve used by SmartHoldem DPoS for transaction signatures (legacy bip-schnorr scheme). Distinct from Ed25519 (used by Iroh) and X25519 (used for E2EE key exchange).

Sled. A high-performance embedded key-value database written in Rust. Used by the Netfory tracker and client for persistent storage of heartbeats, bandit weights (`swarm:weights`), DHT caches, and application state.

Smarthoshi. The smallest indivisible unit of STH (1 STH = 10^8 smarthoshi, analogous to satoshi in Bitcoin). All internal amounts are stored as integers in smarthoshi.

STH. The native utility token of the SmartHoldem / Netfory ecosystem. Used for network fees, identity registration, dApp publication, paid messaging, seeder rewards, and governance. Total supply capped at ~249.7M (post-emission halt).

Swarm Loading. A multi-peer download strategy where the client simultaneously probes multiple seeders and uses the Multi-Armed Bandit algorithm to select the fastest provider. Includes 8-second failover timeouts.

Tauri v2. A framework for building cross-platform desktop applications (Windows, macOS, Linux) using a native WebView for the frontend and Rust for the backend. The foundation of the SmartNet client.

Tracker. A lightweight coordination server (not a content server) that maintains the registry of dApps, seeder heartbeats, uptime scores, and boost-pool weights. Multiple trackers operate in parallel; failure of one does not break the network.

Trackerless Discovery. The ability to find dApp seeders without contacting any HTTP tracker, using Mainline DHT announcements and signed identity-beacons. The tracker becomes a bootstrap accelerator, not a gatekeeper.

u:// / .sth / sn:// / dev:// / sth://. The five URI schemes of the Netfory ecosystem: `u://` (user identity), `.sth` (human-readable domain), `sn://` (internal service routing), `dev://` (source code repositories), `sth://` (dApp WebView origin).

Web 4.0 (Symbiotic Web). The evolutionary stage of the internet where AI agents, IoT devices, and decentralized infrastructure converge into an autonomous, self-sustaining ecosystem. Netfory is a foundational infrastructure layer for Web 4.0.

WebSocket. A full-duplex communication protocol over a single TCP connection. In Netfory, tunneled through Iroh/QUIC to `netfory-provider` nodes, enabling real-time dApp features (poker, chat) without public domains.

X25519. An elliptic-curve Diffie-Hellman key exchange algorithm used in Netfory's end-to-end encrypted messaging for deriving shared secrets between communicating parties.