Trust-Weighted Gossip for Decentralized Storage and Retrieval

A Protocol for Returning Information Custody to the Social Graph

Leon Acosta
leon@nostr-wot.com

March 2026

Abstract

Current decentralized social protocols depend on relay servers that control what gets stored, served, and censored. This paper presents a gossip-based storage and retrieval protocol that returns data custody to the social graph itself. Users form reciprocal *storage pacts* with trust-weighted peers, creating a distributed storage mesh that mirrors how human communities naturally preserve and transmit information. We describe the pact formation mechanism, a tiered retrieval protocol with cascading fallback paths, and a Web of Trust (WoT)-filtered gossip layer that bounds propagation while maintaining epidemic delivery guarantees. Simulation results across a 5,000-node network with realistic follow-to-pact ratios show 96.9% retrieval success with 84.5% of reads served from local pact storage, 12.2% resolved via relay fallback, and only 3.1% failing—validating that a social-graph-first architecture can achieve high availability without centralized infrastructure. We further describe how integration with FIPS (Free Internetworking Peering System) extends the protocol to operate across heterogeneous transports including mesh radio, Bluetooth, and overlay networks, eliminating dependence on the internet itself.

1 Introduction

1.1 The Gossip Parallel

Human communities have always propagated information through gossip. A person shares news with their close circle, who share it with theirs, creating epidemic spread through a trust-weighted social graph. This mechanism has three properties that formal gossip protocols seek to replicate:

1. **Trust filtering**—information from a close friend carries more weight than from a stranger. A claim about person X means different things coming from someone 1 hop versus 4 hops away.
2. **Contextual preservation**—gossip within a community preserves context: who said what, to whom, under what circumstances. Gossip without context is slander; with context it is signal.
3. **Natural redundancy**—important information reaches you through multiple independent paths. If one friend is unavailable, another provides the same update. The redundancy is proportional to the information's social relevance.

These properties map directly onto the protocol primitives we describe: WoT-filtered forwarding (Section 3), volume-matched storage pacts (Section 4), and multi-path retrieval with cascading fallback (Section 5).

1.2 The Relay Problem

In existing decentralized social protocols such as Nostr [1], relays act as the primary infrastructure for event storage, retrieval, and distribution. While relay operators are independent, they collectively form a dependency layer with significant power:

- **Storage control**—relays decide which events to store and for how long.
- **Distribution control**—relays decide which events to serve and to whom.
- **Censorship capability**—relay operators can silently drop events.
- **Economic leverage**—users must pay relays or accept their terms.

This recreates the platform dependency that decentralized protocols were designed to eliminate. The user's social graph lives on infrastructure they do not control.

1.3 Contribution

We propose a local-first architecture where:

- **Primary storage** resides on the user's device and their close WoT peers (1–2 hops).
- **Primary retrieval** queries the trust network first, not relay infrastructure.
- **Relay role** is demoted to optional discovery and curation—useful for beyond-graph reach, not required for existence.

This is gossip protocol in the computer science sense—epidemic information spreading through a peer mesh [2]—with WoT determining propagation priority. The combination of Nostr's key model, social-graph storage, and trust-weighted routing creates a system where the network's social structure *is* its infrastructure.

The paper makes three contributions:

1. A **storage pact mechanism** with volume-balanced reciprocity, challenge-response verification, and popularity-scaled redundancy (Section 4).
2. A **tiered retrieval protocol** with four delivery paths—local, cached endpoint, gossip, and relay fallback—evaluated through discrete-event simulation (Section 5).
3. A **FIPS integration architecture** that extends the protocol to operate over mesh radio, Bluetooth, serial links, and other transports without internet dependence (Section 7).

2 Communities and Protocols: A Structural Parallel

The design of this protocol is not merely inspired by human social dynamics—it is structurally isomorphic to them. Each protocol primitive maps to an observable pattern in how humans form communities, maintain relationships, and propagate information.

2.1 Inner Circle, Outer Circle, Acquaintances

Human social networks exhibit concentric structure. Robin Dunbar's research [3] identifies layers: ∼5 intimate contacts, ∼15 close friends, ∼50 good friends, ∼150 casual friends, with each layer roughly tripling in size and halving in intimacy.

The protocol mirrors this with three node roles:

Full nodes are the protocol's equivalent of the friends who remember everything—your complete history is safe with them. Light nodes are the broader social circle: they know what you've

Human Layer	Protocol Equivalent	Persona	Storage Obligation	Uptime
Inner circle (5–15)	Full nodes (25%)	*Keeper*	Complete event history	95%
Extended circle (50–150)	Light nodes (75%)	*Witness*	Rolling 30-day window	30%
Acquaintances (150+)	Relay-discovered peers	*Herald*	None (optional caching)	Variable

been up to recently, but don't maintain archives. Acquaintances discovered through relays are the weak ties [4] that provide reach beyond your community, without storage commitment.

2.2 Reciprocity as Infrastructure

In human communities, relationships survive through reciprocity. You remember my stories; I remember yours. One-sided relationships decay. The protocol formalizes this as *storage pacts*: bilateral agreements where both parties store each other's events, matched by data volume within a 30% tolerance.

This volume matching is the protocol equivalent of activity-matched friendships. A prolific poster (high volume) paired with a lurker (low volume) creates an asymmetric obligation that incentivizes defection—just as asymmetric friendships decay in real social networks. The protocol prevents this by matching peers with compatible activity levels.

2.3 Reputation Through Behavior, Not Scores

Human reputation is not a number. It is the aggregate of observed behavior, weighted by the observer's trust in each source. The protocol's reliability scoring operates identically:

- Each node maintains **per-peer reliability scores** using a 30-day rolling window of challenge-response results.
- Scores are private—each node computes its own assessment.
- A score above 90% is healthy; between 50–70% triggers replacement; below 50% means immediate drop.
- Failed peers naturally lose their reciprocal storage—the same way unreliable friends are gradually excluded from information flow.

There is no global reputation score. The network topology *is* the incentive structure. A node with 20 reliable pact partners has 20 advocates forwarding its content through gossip. A dropped pact means a lost advocate and reduced reach—the protocol equivalent of being talked about less.

2.4 Guardianship—Bootstrapping Through Generosity

Every community has patrons—established members who vouch for newcomers they don't personally know. A shopkeeper who gives a stranger their first job. A neighbor who co-signs a lease for a new arrival. These acts of generosity bootstrap trust where none exists.

The protocol formalizes this as *guardian pacts*: an established user (*Guardian*) voluntarily stores data for one untrusted newcomer (*Seedling*) outside their Web of Trust. Unlike bootstrap pacts (triggered by a follow), guardian pacts are volunteered—the Guardian opts in, accepting a small storage cost to help the network grow.

	Bootstrap Pact	**Guardian Pact**
Trigger	Seedling follows someone	Guardian volunteers
WoT required	No (follow creates edge)	No
Capacity	One per followed user	One per Guardian
Expiry	90 days or 10 reciprocal pacts	90 days or Hybrid phase (5+ pacts)
Reciprocity	One-sided	One-sided

The pay-it-forward framing is deliberate: today's Seedling becomes tomorrow's Guardian. A user who was supported during their bootstrap phase is encouraged (by client UX, not protocol enforcement) to volunteer a guardian slot once they reach Sovereign phase.

2.5 Gossip as Curated Propagation

When humans gossip, they don't broadcast to everyone. They share selectively based on trust and relevance. The protocol's WoT-filtered forwarding mirrors this:

- **Active pact partners** (highest priority)—you forward eagerly for those whose data you store.
- **1-hop WoT peers**—people you directly follow. You'll pass along their content.
- **2-hop WoT peers**—friends-of-friends. You'll forward if capacity permits.
- **Unknown sources**—never forwarded. Strangers' gossip stops at your door.

This creates a natural boundary for information propagation that matches Dunbar's social brain hypothesis [3]: the protocol's gossip radius is bounded by the same trust gradient that bounds human social information flow.

3 Protocol Primitives

3.1 Identity and Trust

Each participant holds a secp256k1 keypair, compatible with Nostr's identity model [1]. The public key serves as both identity and address. A key hierarchy isolates device-level operations from identity-level authority:

- **Root key** (cold storage)—signs device delegations only.
- **Governance key**—signs profile and follow list updates.
- **Device subkeys**—sign day-to-day events (posts, reactions, DMs).

The Web of Trust is defined by follow relationships. A node p follows a set of nodes F_p. The WoT distance $d(p, q)$ is the minimum number of follow-hops from p to q. The protocol operates within a 2-hop boundary: nodes forward gossip only to peers where $d(p, q) \leq 2$.

3.2 Node Types

Let $N = \{n_1, n_2, \dots, n_k\}$ be the set of network participants. Each node n_i has type $t_i \in \{\text{Full}, \text{Light}\}$:

- **Full nodes** (*Keepers*) maintain complete event history for their pact partners. Expected uptime: $u_{\text{full}} = 0.95$.
- **Light nodes** (*Witnesses*) maintain events within a checkpoint window W (default 30 days) for their pact partners. Expected uptime: $u_{\text{light}} = 0.30$.

The network composition follows the distribution observed in real deployments: approximately 25% Full nodes (always-on servers, dedicated hardware) and 75% Light nodes (mobile devices, intermittent connectivity).

3.3 Events

An event $e = (\text{id}, \text{author}, \text{kind}, \text{size}, \text{seq}, \text{prev\_hash}, \text{created\_at})$ is the atomic unit of information. Events are cryptographically signed by the author's device key and are immutable once published.

The weighted average event size under the default content mix is:

Formula F-01:

$$\mathbb{E}[\text{size}] = \sum_k \text{size}_k \cdot \text{mix}_k = 800(0.40)+500(0.30)+600(0.15)+900(0.10)+5500(0.05) = 925 \text{ bytes} \tag{1}$$

3.4 Checkpoints

Checkpoints (kind 10051) are periodic reconciliation markers published by each user. A checkpoint contains:

- Per-device event heads (latest event ID and sequence number per device).
- A Merkle root over all events since the previous checkpoint.
- References to current profile and follow list.

Checkpoints enable light nodes to verify data completeness without downloading full history, and define the storage obligation boundary for light pact partners.

4 Storage Pact Mechanism

4.1 Pact Formation

A storage pact is a bilateral agreement between two nodes to store each other's events. Formation follows a three-phase DVM-style (Data Vending Machine) negotiation:

1. **Request** (kind 10055): Node p broadcasts a storage pact request with its volume estimate, minimum pact count, and TTL.
2. **Offer** (kind 10056): Qualifying WoT peers respond with offers.
3. **Accept** (kind 10053): Both parties exchange private pact events and begin mutual storage.

Qualification requires: WoT membership (follows or followed-by), volume balance within tolerance δ, and minimum account age (default 7 days to prevent Sybil pact formation).

4.2 Volume Balancing

Let V_p and V_q be the data volumes of nodes p and q. A pact is balanced when:

$$\frac{|V_p - V_q|}{\max(V_p, V_q)} \leq \delta \tag{2}$$

where $\delta = 0.30$ (30% tolerance). This ensures symmetric risk: neither partner bears disproportionate storage cost.

4.3 Pact Topology

Each node maintains two classes of pact partners:

- **Active pacts** (default: 20): Partners that are regularly challenged and expected to serve data on request.
- **Standby pacts** (default: 3): Partners that receive events but are not challenged. Standby partners are promoted to active when an active partner fails, providing instant failover without renegotiation delay.

For high-follower accounts, the pact count scales:

Followers	Active Pacts
< 100	10
100–1,000	20
1,000–10,000	30
10,000+	40+

4.4 Proof of Storage

Pact partners are verified through periodic challenge-response exchanges (kind 10054):

Hash challenge: The challenger specifies a range of event sequence numbers and a nonce. The partner computes $H(\text{events}[i..j] \| \text{nonce})$ and returns the hash. This proves possession without transferring full events.

Serve challenge: The challenger requests a specific event by sequence number and measures response latency. Consistently slow responses (>500ms) suggest the partner is proxying rather than storing locally. Flagged peers receive 3× challenge frequency.

4.5 Reliability Scoring

Each node maintains per-peer reliability scores using an exponential moving average:

$$\text{score}' = \text{score} \cdot \alpha + \text{result} \cdot (1 - \alpha) \tag{3}$$

where $\alpha = 0.95$ and result $\in \{0, 1\}$. This gives a 30-day effective window with recent challenges weighted more heavily.

Score	Status	Action
$\geq 90\%$	Healthy	No action
70–90%	Degraded	Increase challenge frequency
50–70%	Unreliable	Begin replacement negotiation
$< 50\%$	Failed	Drop immediately, promote standby

4.6 Storage Obligations

The total storage obligation per user depends on the pact partner composition:

Formula F-03 (storage per user):

$$S = P \cdot (f \cdot \mathbb{E}[\text{size}] \cdot R \cdot D_{\text{full}} + (1 - f) \cdot \mathbb{E}[\text{size}] \cdot R \cdot D_{\text{light}}) \tag{4}$$

where P = number of active pacts (default 20), f = fraction of Full node partners (~0.25), $\mathbb{E}[\text{size}] = 925$ bytes, R = events per day (default 25), D_{full} = complete history duration, $D_{\text{light}} = \min(\text{duration}, W)$.

4.7 Guardian Pacts

Guardian pacts extend the pact mechanism to support newcomers who have no Web of Trust presence. An established user (a *Guardian*) volunteers to store data for one newcomer (a *Seedling*) without WoT membership or volume matching requirements.

Guardian pacts use the same kind 10053 event with a `type: guardian` tag. Formation flow:

1. **Opt-in**: A Sovereign-phase node advertises guardian availability via kind 10055 with `type: guardian`.
2. **Match**: A Seedling with fewer than 5 pacts is matched (client-side or relay-assisted).
3. **Accept**: Both parties exchange kind 10053 with `type: guardian`.
4. **Store**: The Guardian stores the Seedling's events. Challenge-response verification (kind 10054) applies.

Expiry conditions: 90 days from formation, or the Seedling reaches Hybrid phase (5+ reciprocal pacts). Each Guardian holds at most one active guardian pact, bounding the storage cost to a single user's data volume.

4.8 Simulation Results: Storage

In a 100-node simulation over 30 days (seed 42, deterministic):

Metric	Value
Mean pact count per node	12.72
Total pacts formed	1,460
Total pacts dropped	53
Churn rate	1.8%
F-18: Full pact fraction (expected)	0.25
F-18: Full pact fraction (actual)	0.25

The full pact fraction matches the network's full node percentage exactly, confirming that pact formation reflects network composition without bias.

5 Retrieval Protocol

5.1 Delivery Tiers

When a node needs to retrieve events from a followed author, the protocol attempts delivery through a cascade of increasingly expensive paths:

Tier 0—BLE mesh (nearby): Nearby devices serve events via Bluetooth Low Energy mesh, relayed up to 7 hops. No internet required. Interoperable with FIPS transport layer (Section 7). Latency: variable.

Tier 1—Instant (local): The node already stores the author's events locally, either through an active pact or from a previous read cache. Cost: zero network traffic. Latency: zero.

Tier 2—Cached Endpoint: The node has cached endpoint addresses (kind 10059) for the author's storage peers. A direct connection retrieves the events without gossip overhead. Latency: ~60ms base + 20ms jitter.

Tier 3—Gossip (kind 10057): The node broadcasts a blinded data request to its WoT peers with TTL=3. The request uses a blinded pubkey $bp = H(\text{target\_pubkey}\|\text{YYYY-MM-DD})$ that rotates daily. Latency: ∼80ms per hop + 30ms jitter per hop.

Tier 4—Relay Fallback: After a configurable timeout (default 30s), the node falls back to a traditional relay query. This is the path of last resort. Latency: ∼200ms base + 50ms jitter.

Each successive tier is attempted only when the previous tier fails or times out, creating a natural cost gradient that keeps most traffic within the social graph.

5.2 Read Cache and Cascading Replication

A critical property emerges from the retrieval protocol: **reads create replicas**. When Bob fetches Alice's events from a storage peer, Bob now holds a local copy. When Carol subsequently requests Alice's events via gossip, Bob can respond—without being one of Alice's formal pact partners.

This creates cascading read-caches that replicate popular content across the follower base:

1. Alice's 20 pact partners hold her events.
2. Each of Alice's followers who reads her events becomes an informal cache.
3. Subsequent readers find the data closer in the social graph.
4. Load scales with $O(\text{followers})$, not $O(\text{pact\_partners})$.

The read cache is bounded (default 100MB) and LRU-evicted.

5.3 Blinded Requests and Privacy

Data requests (kind 10057) use blinded pubkeys to preserve reader privacy:

$$bp = H(\text{target\_root\_pubkey}\|\text{YYYY-MM-DD}) \tag{5}$$

The blinding salt rotates daily. Peers match incoming requests against both today's and yesterday's salts to handle clock skew. This ensures:

- Storage peers learn that *someone* requested the data, but not *who*.
- Observers cannot link requests across days.
- The requesting node's reading patterns are not exposed to the gossip network.

5.4 Gossip Reach Analysis

With a mean degree of 20 peers and TTL=3, the gossip reach at each hop (accounting for 25% clustering coefficient) is:

```
hop 1: 20 nodes
hop 2: 20 * 20 * (1 - 0.25) = 300 nodes
hop 3: 300 * 20 * (1 - 0.25) = 4,500 nodes
```

Cumulative reach: ∼4,820 nodes within 3 hops.

5.5 Rate Limiting and Gossip Hardening

The gossip layer implements per-source rate limiting to prevent amplification attacks:

- Kind 10055 (pact requests): 10 req/s per source pubkey.
- Kind 10057 (data requests): 50 req/s per source pubkey.

Rate limiting uses a sliding window counter per source. Excess requests are silently dropped. Combined with WoT-only forwarding, this bounds the gossip blast radius to the trust network while maintaining epidemic delivery within it.

Request deduplication uses an LRU cache of 10,000 request IDs. Duplicate requests are dropped silently, preventing gossip loops and reducing amplification.

5.6 Data Availability

The probability that all pact partners are simultaneously offline determines data unavailability:

Formula F-14 (all-pacts-offline probability):

$$P(\text{unavailable}) = (1 - u_{\text{full}})^{P \cdot f} \cdot (1 - u_{\text{light}})^{P \cdot (1-f)} \tag{6}$$

With default parameters ($P = 20$, $f = 0.25$, $u_{\text{full}} = 0.95$, $u_{\text{light}} = 0.30$):

$$P(\text{unavailable}) = (0.05)^5 \cdot (0.70)^{15} = 3.125 \times 10^{-7} \cdot 4.747 \times 10^{-3} = 1.48 \times 10^{-9} \tag{7}$$

This represents approximately one-in-a-billion chance of complete data unavailability at any instant—comparable to enterprise storage system reliability, achieved entirely through social graph redundancy.

5.7 Simulation Results: Retrieval

Results from a 5,000-node, 30-day deterministic simulation (BA $m = 50$, seed 42):

Metric	**Value**
Total read attempts	1,733,380
Overall success rate	96.9%
Tier 1 (Instant/local)	1,465,527 (84.5%)
Tier 2 (Cached endpoint)	0 (0.0%)
Tier 3 (Gossip)	3,322 (0.2%)
Tier 4 (Relay fallback)	211,509 (12.2%)
Failed	53,022 (3.1%)
Paths tried per request (p50)	1.0
Paths tried per request (p95)	3.0
Paths tried per request (p99)	3.0
Paths tried per request (mean)	1.31

The dominant result remains Tier 1: 84.5% of reads find data locally via pact partnerships. This is a significant drop from the 100-node run's 95.8% (which was inflated by graph density), confirming that realistic follow-to-pact ratios produce meaningful tier diversity. The relay fallback path (Tier 4) handles 12.2% of reads—substantially more than the 100-node run's 2.7%, reflecting the realistic scenario where most followed accounts are not pact partners. Gossip

(Tier 3) resolves only 0.2% of reads, while the cached endpoint tier (Tier 2) sees no traffic in this simulation.

The 3.1% failure rate represents reads where all paths (gossip + relay) failed, typically when the target author's pact partners were offline and relay fallback did not resolve within the 30-second timeout.

Note: The simulation does not yet model BLE mesh transport (Tier 0). Retrieval results reflect Tiers 1–4 only.

5.8 Latency Distribution

Tier	**p50 (ms)**	**p95 (ms)**	**p99 (ms)**
Tier 1 (Instant)	0	0	0
Tier 3 (Gossip)	∼160	∼320	∼480
Tier 4 (Relay)	30,200	30,200	30,200

Relay latency includes the 30-second gossip timeout before fallback, making it a last-resort path. In the 5,000-node simulation, 12.2% of reads reach Tier 4—a realistic penalty for reads targeting authors outside the reader's pact set. The protocol is designed so that most reads (84.5%) avoid this penalty entirely through local pact storage.

6 Flow Control

6.1 The Capacity Problem

Van Renesse et al. [2] demonstrated that anti-entropy protocols have bounded capacity: under high update load, gossip messages cannot carry all required deltas, and update latency grows without bound. Our protocol faces an analogous constraint: each node's gossip bandwidth is finite, and the rate of storage pact requests, data requests, and event deliveries must be controlled.

6.2 Pact-Aware Priority

Rather than the Scuttlebutt reconciliation's approach of ordering deltas by version number, our protocol orders gossip forwarding by social proximity:

1. **Active pact partners**: forwarded immediately, no queuing.
2. **1-hop WoT**: forwarded with standard priority.
3. **2-hop WoT**: forwarded when capacity is available.
4. **Beyond WoT**: never forwarded.

This is analogous to scuttle-depth ordering [2]—the protocol prioritizes propagating information for the peers most relevant to the local node, rather than being "fair" across all participants.

6.3 Three-Phase Adoption

The protocol includes a built-in flow control mechanism through its adoption phases:

New users begin in Bootstrap phase with full relay dependence. As they form pacts, traffic gradually shifts from relays to peer mesh. This provides natural flow control: the rate at which a new node joins the gossip network is limited by the rate at which it forms pacts, preventing gossip overload from sudden influx.

Phase	Pact Count	Behavior
Bootstrap	0–5	Publish to relays primarily; form pacts as available
Hybrid	5–15	Publish to both relays and peers; fetch from peers first
Sovereign	15+	Storage peers primary; relays optional accelerator

6.4 Pact Renegotiation Jitter

When a user's activity changes and pact renegotiation is needed, the protocol introduces random jitter (0–48 hours) before broadcasting replacement requests. Standby pacts provide immediate failover during this delay. This prevents renegotiation storms—the gossip equivalent of TCP's synchronized loss recovery problem [5].

7 FIPS Integration: Beyond the Internet

7.1 Motivation

The protocol as described assumes IP connectivity between nodes. This creates a residual infrastructure dependency: the internet itself. FIPS (Free Internetworking Peering System) [6] eliminates this dependency by providing a self-organizing mesh network that operates natively over heterogeneous transports.

7.2 FIPS Architecture

FIPS implements three protocol layers:

Transport Layer: Delivers datagrams over arbitrary media—UDP/IP, Ethernet, Bluetooth Low Energy, serial links, radio modems, Tor circuits.

FIPS Mesh Protocol (FMP): Authenticates peers via Noise IK handshakes, builds a self-organizing spanning tree for coordinate-based routing, and propagates reachability via bloom filters.

FIPS Session Protocol (FSP): Provides end-to-end authenticated encryption via Noise XK handshakes. Sessions are bound to Nostr keypairs (npubs), not transport addresses, so they survive route changes and transport switching.

7.3 Identity Convergence

FIPS uses Nostr keypairs (secp256k1) as native node identities. This is the critical integration point: **a Gozzip user's root pubkey is also their FIPS network address**. No bridging, translation, or identity mapping is required. A storage pact partner is simultaneously a mesh routing peer.

The FIPS node_addr (a 16-byte SHA-256 hash of the pubkey) serves as the routing identifier in packet headers, while the npub serves as the application-layer address. Both are deterministically derived from the same keypair.

7.4 Transport Independence for Gossip

With FIPS as the transport layer, the gossip protocol operates identically regardless of the underlying medium:

Scenario	Transport Path	Gossip Behavior
Both nodes on internet	UDP/IP overlay	Standard gossip
One node on local mesh	BLE + WiFi	Gossip via mesh relay
Both nodes offline	BLE mesh (7 hops)	Local gossip, store-and-forward
Censored network	Tor transport	Gossip via onion routing
Remote/rural	Radio modem	Low-bandwidth gossip

7.5 Spanning Tree Meets Social Graph

FIPS builds a spanning tree for coordinate-based routing using distributed parent selection. Each node chooses a parent based on measured link quality (RTT, loss, jitter), creating a tree that reflects physical network topology.

The social graph's WoT structure overlays this physical topology. In many cases, WoT peers will also be FIPS mesh peers—a followed user's always-on full node is likely to be configured as a direct FIPS peer. Where social and physical topology diverge, FIPS's bloom filter discovery provides the bridge.

7.6 Offline-First Operation

FIPS enables a genuinely offline-first mode:

1. **BLE mesh transport** (Layer 0 in the delivery priority): nearby devices relay events up to 7 hops via Bluetooth Low Energy, encrypted with Noise Protocol XX handshakes.
2. **Store-and-forward queuing**: when no transport is available, events are queued locally (up to 1,000 events or 50MB) and auto-drain when any transport becomes available.
3. **Geohash discovery**: ephemeral subkeys (not linked to identity) with geohash tags enable nearby-device discovery without revealing identity.

7.7 The "Pub Server" Problem, Solved Differently

Scuttlebutt [7] identified the fundamental tension in peer-to-peer social networks: mobile devices go offline, but content must remain available. Scuttlebutt's solution was "pub servers"—always-on nodes that replicate data. But pub servers are relays by another name.

Our protocol addresses this through three mechanisms:

1. **Full nodes as pact partners**: The 25% of network participants that are always-on serve as full-history storage peers. Unlike pub servers, they have bilateral obligations enforced by challenge-response.
2. **Heterogeneous transport fallback**: When a mobile device goes offline on cellular, it may still be reachable via BLE mesh. FIPS routing finds the path.
3. **Standby pact promotion**: When a pact partner goes unreachable, a standby partner is immediately promoted—no renegotiation, no discovery delay.

8 Related Work

8.1 Anti-Entropy and Epidemic Protocols

The foundational work on epidemic algorithms for replicated database maintenance [8] established that updates spread in $O(\log N)$ rounds through random peer selection. Van Renesse et al. [2] extended this with the Scuttlebutt reconciliation mechanism and AIMD-based flow control. Our protocol adopts the epidemic spreading model but replaces random peer selection with WoT-weighted selection, sacrificing theoretical convergence speed for trust-bounded propagation.

8.2 Scuttlebutt / Secure Scuttlebutt

Secure Scuttlebutt (SSB) [7] proved the viability of gossip-based social networking with local-first storage and append-only feeds. Our protocol builds on SSB's core insight but differs in three ways:

1. **Nostr key model**: SSB uses ed25519 feeds tied to devices; we use secp256k1 keypairs with a delegation hierarchy that supports multi-device and key rotation.
2. **Bilateral pacts vs. unilateral replication**: SSB's pubs replicate unilaterally; our pacts enforce reciprocal obligation with proof of storage.
3. **WoT-bounded gossip**: SSB's gossip has no explicit trust boundary; our protocol restricts forwarding to 2-hop WoT distance.

8.3 Proof of Storage

The challenge-response proof of storage mechanism draws from Filecoin's Proof of Replication and Proof of Spacetime [9], and Arweave's Succinct Proofs of Random Access [10]. Our protocol uses simpler primitives because the threat model is different: pact partners are WoT peers with social incentive to maintain the relationship, not anonymous miners requiring cryptoeconomic guarantees.

8.4 FIPS and Mesh Networking

FIPS [6] provides the transport-agnostic mesh layer that our protocol requires for internet-independent operation. FIPS's use of Nostr keypairs for node identity creates a natural integration point. The spanning tree routing with bloom filter discovery is complementary to our WoT-based gossip routing: FIPS handles physical reachability while the gossip layer handles social relevance.

8.5 Local-First Software

The Ink & Switch research group's work on local-first software [11] articulated the principles our protocol implements: data ownership, offline operation, and collaboration without mandatory servers. Our contribution is extending these principles to a social networking context where the "collaborators" are an entire social graph, not a small team.

8.6 Infrastructure Benchmark: Nostr and Mastodon

The two most prominent decentralized social protocols—Nostr [1] and Mastodon/ActivityPub [12]—represent fundamentally different architectural choices. Nostr separates identity from infrastructure but depends on relay servers. Mastodon binds identity to infrastructure through domain-coupled accounts. Our protocol makes the social graph itself the infrastructure. The following tables provide a structured comparison.

	Nostr	**Mastodon**	**Gozzip**
Identity	secp256k1 keypair	`user@instance` (domain-bound)	secp256k1 + key hierarchy
Portability	Full—identity is the keypair	Partial—followers migrate; posts do not	Full—same Nostr key model with delegation
Multi-device	Share private key or NIP-26	Native—server holds session	Root key delegates to device subkeys
Recovery	None—lose key, lose identity	Admin can reset password	Root key revokes subkeys; root loss is fatal

	Nostr	**Mastodon**	**Gozzip**
Where data lives	Relay servers	Home instance (PostgreSQL)	User's device + ∼20 pact partners
Who controls	Relay operators	Instance admin	User + bilateral pact partners
Redundancy	Manual multi-relay publishing	None—single home instance	20 active + 3 standby pacts; P(all-offline) $\approx 10^{-9}$
If infra dies	Events lost unless duplicated	All posts from instance lost	Pact partners retain copies

Identity and Portability

Data Storage and Ownership

Content Distribution

Censorship Resistance

Scalability and Costs

Architectural Summary

Nostr's design cleanly separates identity from infrastructure, but relays remain the storage and distribution layer—recreating a dependency surface analogous to the platforms it replaces. The outbox model (NIP-65) and negentropy sync (NIP-77) improve relay coordination but do not eliminate relay dependence.

Mastodon bundles identity with infrastructure. The `user@instance` model means your account's survival depends on your instance operator's continued goodwill, funding, and uptime. FEP-ef61 (portable objects) and FEP-8b32 (object integrity proofs) aim to decouple identity from instances, but these remain proposals.

Our protocol addresses both failure modes by making the social graph the infrastructure layer. Storage pacts distribute data custody across WoT peers with cryptographic verification. The relay is demoted from gatekeeper to optional fallback. The trade-off is protocol complexity: pact negotiation, challenge-response verification, WoT computation, and multi-tier retrieval are substantially more complex than Nostr's REQ/EVENT or Mastodon's HTTP POST delivery.

The honest gap: our protocol is validated by simulation (100–5,000 nodes). Nostr has ∼1,000 relays and real users. Mastodon has ∼26,000 instances and ∼1.2M monthly active users. The architectural claims require production-scale validation.

	Nostr	Mastodon	Gozzip
Mechanism	Client pulls via WebSocket	Server pushes via HTTP POST	Tiered cascade: local → endpoint → gossip → relay
Discovery	Author's outbox relays (NIP-65)	Federated timeline, hashtags	WoT gossip with blinded pubkeys
Read privacy	Relay sees all subscriptions	Admin sees all activity; DMs unencrypted	Blinded pubkeys rotate daily
Offline	Not supported	Not supported	BLE mesh via FIPS; store-and-forward

	Nostr	Mastodon	Gozzip
Surface	Individual relays can drop events	Admin has unilateral suspend/delete	No single point—data across 20+ WoT peers
Filtering	Per-relay policies; NIP-42; PoW	Per-instance moderation; domain blocks	WoT filtering; 2-hop trust boundary
Account risk	Low (self-sovereign key)	High—admin can suspend	Very low—bilateral obligation

9 Evaluation Summary

9.1 Simulation Environment

We evaluate the protocol using a discrete-event simulator written in Rust. The simulator models individual nodes as async actors communicating through message-passing channels, with a central router providing configurable per-path latency distributions.

Default configuration:

- 100 nodes (25 Full, 75 Light)
- Barabási–Albert graph (10 edges per node, seed 42)
- 30-day simulation, 60-second tick interval
- Deterministic mode for reproducibility

9.2 Formula Validation

100-node run:

5,000-node run:

Note on F-03, F-14, and F-31: F-03 (storage per user) fails at both scales because not all nodes fill their pact quotas to capacity over the simulation duration—the formula assumes sustained maximum output from all pact partners. F-14 improves dramatically at 5k scale (deviation drops from 10^6 to 277%), approaching the theoretical prediction as the network grows. F-31 (gossip rate) shows a large deviation because the formula models steady-state gossip while the simulator generates bursty gossip traffic from active sessions. The three passing formulas (F-01, F-24, F-18) validate the core protocol mechanics: event sizing, uptime modeling, and pact composition.

9.3 Per-Node Metrics

100-node run:

	Nostr	Mastodon	Gozzip
Cost bearer	Relay operators ($5–20/mo paid relays)	Instance operators (10–20 GB/day media cache)	Distributed; volume-balanced within 30%
Bandwidth	Variable; duplicates across relays	One POST per remote instance	~11.7 MB/day (Full); ~4.1 MB/day (Light) at 5k scale
Popular content	Each relay serves independently	Boosts replicate per instance	Cascading read-caches: O(followers)
Infrastructure	~1,000 relays	~26,000 instances	25% Full / 75% Light; no central infra

Formula	Description	Expected	Actual	Deviation	Status
F-01	Avg event size	925 B	925 B	0.0%	Pass
F-24	Online fraction	0.4625	0.4622	0.07%	Pass
F-18	Full pact fraction	0.25	0.25	0.0%	Pass
F-03	Storage per user	13.9 MB	527 KB	96.2%	(see note)
F-14	All-offline prob.	1.48×10^{-9}	2.49×10^{-6}	—	(see note)

5,000-node run:

At 5,000 nodes, bandwidth requirements scale significantly: Full nodes average 11.68 MB/day and Light nodes 4.14 MB/day. This is driven by the higher follow count (BA $m = 50$) producing more gossip traffic and larger pact storage obligations. These remain within typical mobile data budgets (~350 MB/month for Full, ~124 MB/month for Light). Pact counts nearly reach the target of 20 (mean 18.69), a substantial improvement over the 100-node run's 12.72, confirming that larger networks better satisfy the pact formation algorithm.

9.4 Large-Scale Validation (5,000 nodes)

The 100-node results above exhibit an artifact of small, dense graphs: with BA $m = 10$ on 100 nodes, each node's ~10 follows nearly coincide with their ~12.72 pact partners, producing 95.8% Tier 1 (instant) reads. Real social networks have a much wider follow-to-pact ratio—users follow 50–500 accounts but maintain only ~20 storage pacts.

To validate under realistic conditions, we run a 5,000-node simulation with BA $m = 50$:

With ~50 follows but only ~20 pacts, roughly 60–80% of followed accounts are *not* pact partners. Reads from those accounts must traverse gossip or relay paths, exercising the full retrieval tier stack.

Retrieval tier distribution:

The 5,000-node results confirm the architectural hypothesis: Tier 1 drops from the artificially inflated 95.8% to a realistic 84.5%, while relay fallback (Tier 4) rises from 2.7% to 12.2%. This reflects the intended design—most reads are served locally through pact partnerships, but a meaningful fraction of reads for non-pact followed accounts requires relay assistance.

The 3.1% failure rate at 5k (vs. 0.7% at 100 nodes) reflects the wider gap between follow sets and pact sets: when a followed author has no online pact partners and the relay fallback also fails within the 30-second timeout, the read is marked as failed.

Pact formation at scale:

Pact formation improves significantly at scale: the mean pact count reaches 18.69 (93.5% of the 20-pact target), compared to 12.72 at 100 nodes. The higher churn rate (6.2% vs. 1.8%) reflects

Formula	Description	Expected	Actual	Deviation	Status
F-01	Avg event size	925 B	925 B	0.0%	Pass
F-24	Online fraction	0.4625	0.4620	0.11%	Pass
F-18	Full pact fraction	0.25	0.2492	0.32%	Pass
F-03	Storage per user	13.9 MB	884 KB	93.6%	Fail
F-14	All-offline prob.	1.48×10^{-9}	5.60×10^{-9}	277%	Fail
F-31	Gossip rate/node	2.89×10^{-6}	0.035	>1M%	Fail

Metric	p50	p95	p99	Mean
Data availability	0.30	0.95	0.95	0.46
Bandwidth (Full, MB/day)	0.076	0.169	0.196	0.090
Bandwidth (Light, MB/day)	0.025	0.059	0.067	0.030
Gossip received/node	909	1,266	1,317	916
Pact count	11	20	20	12.72

the larger, more dynamic network but remains manageable with 3 standby pacts per node.

10 Conclusion

We have presented a gossip-based storage and retrieval protocol that returns data custody from relay infrastructure to the social graph. The protocol's design mirrors human social dynamics: reciprocal storage pacts parallel reciprocal friendships, WoT-filtered gossip parallels trust-based information sharing, and the Full/Light node distinction mirrors the inner-circle/outer-circle structure of human communities.

Simulation results at 5,000 nodes validate the approach: 96.9% retrieval success with 84.5% of reads served from local pact storage and 12.2% resolved via relay fallback demonstrates that high data availability is achievable through social graph redundancy, with relays serving as an effective but non-critical safety net. The relay is demoted from gatekeeper to optional accelerator—useful for reads targeting non-pact followed accounts, but not required for existence.

Integration with FIPS extends the protocol beyond internet dependence, enabling operation over mesh radio, Bluetooth, and other transports. The shared Nostr identity model eliminates bridging complexity: a user's social identity is simultaneously their network address.

The honest assessment: this architecture does not eliminate the need for always-on infrastructure. Full nodes—*Keepers*—(25% of the network) are the protocol's equivalent of reliable friends who are always available. The difference is structural: these nodes operate under bilateral obligations enforced by challenge-response, not under unilateral control of a platform operator. The infrastructure exists, but it is owned by the social graph.

The hard problems remain. Graph bootstrap for new users requires initial relay dependence. The 75/25 Full/Light split must emerge organically through incentives, not be mandated. DM key rotation provides bounded forward secrecy but not perfect. These are engineering challenges, not architectural ones—the social graph is a viable foundation for decentralized infrastructure.

Metric	**p50**	**p95**	**p99**	**Mean**
Data availability	0.30	0.95	0.95	0.46
Bandwidth (Full, MB/day)	11.45	14.47	18.42	11.68
Bandwidth (Light, MB/day)	3.79	6.43	10.14	4.14
Gossip received/node	74,212	175,127	180,690	90,074
Pact count	20	20	20	18.69
Pact churn rate	—	—	—	6.2%

Parameter	**100-node run**	**5,000-node run**
Nodes	100	5,000
BA edges/node	10	50
Follows/node	∼10	∼50
Pacts/node (mean)	12.72	18.69
Duration	30 days	30 days
Reads/day/user	50	50
Total read attempts	34,432	1,733,380
Elapsed (wall clock)	∼9s	11,657s (∼3.2h)

References

[1] Nostr Protocol. "Nostr Implementation Possibilities." https://github.com/nostr-protocol/nips

[2] R. van Renesse, D. Dumitriu, V. Gough, C. Thomas. "Efficient Reconciliation and Flow Control for Anti-Entropy Protocols." LADIS Workshop, 2008.

[3] R. Dunbar. "Neocortex size as a constraint on group size in primates." *Journal of Human Evolution*, 22(6):469–493, 1992.

[4] M. Granovetter. "The Strength of Weak Ties." *American Journal of Sociology*, 78(6):1360–1380, 1973.

[5] V. Jacobson. "Congestion Avoidance and Control." ACM SIGCOMM, 1988.

[6] J. Corgan. "FIPS: Free Internetworking Peering System." https://github.com/jmcorgan/fips

[7] D. Tarr, E. Lavoie, A. Chen, J. Robinson. "Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications." ACM PLDI, 2019.

[8] A. Demers, D. Greene, C. Hauser, et al. "Epidemic Algorithms for Replicated Database Maintenance." ACM PODC, 1987.

[9] Protocol Labs. "Filecoin: A Decentralized Storage Network." 2017.

[10] S. Williams, V. Diiorio, S. Barski. "Arweave: A Protocol for Economically Sustainable Information Permanence." 2019.

[11] M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan. "Local-First Software: You Own Your Data, in Spite of the Cloud." Onward!, 2019.

[12] C. Webber, J. Tallon. "ActivityPub." W3C Recommendation, 2018. https://www.w3.org/TR/activitypub/

Tier	**Description**	**100-node**	**5,000-node**
Tier 1	Instant (pact partner)	95.8%	84.5%
Tier 2	Cached endpoint	—	0.0%
Tier 3	Gossip (WoT hop)	0.8%	0.2%
Tier 4	Relay fallback	2.7%	12.2%
—	Failed	0.7%	3.1%
	Overall success rate	**99.3%**	**96.9%**

Metric	**100-node**	**5,000-node**
Mean pact count	12.72	18.69
Total pacts formed	1,460	115,868
Total pacts dropped	53	9,357
Net active pacts	1,407	106,511
Churn rate	1.8%	6.2%
Full pact fraction	0.25	0.249