SHARDNET: A Serverless, Peer-to-Peer Protocol for Encrypted Data Exchange and Resilient Storage

Shardnet Project

Version 0.2 | 2026-06-23

shardnet.app/whitepaper | github.com/stablediffusion-ai/shard

Contents

Version	Date	Description
0.1	2026-06-23	Initial draft — abstract and high-level sections
0.2	2026-06-23	Full technical specification from source audit

1 Abstract

Shardnet is an application-layer peer-to-peer (P2P) protocol for secure, serverless data storage and exchange. Every file is encrypted client-side with AES-256-GCM, split into fifteen Reed-Solomon fragments, and distributed across the network. Any ten fragments are sufficient to reconstruct the original. Nodes communicate directly over QUIC/TLS 1.3 using a Kademlia-based DHT for discovery and routing. No central server holds data, credentials, or routing state. This document is the authoritative technical specification of the Shardnet protocol: transport, erasure coding, cryptography, identity, storage, and network maintenance.

2 Introduction

2.1 Design Principles

Shardnet is built on four invariants:

1. **No central intermediary.** Every network function — discovery, routing, storage, transfer — is carried out by peers. There is no server that can be seized, blocked, or compelled to reveal user data.
2. **Zero-knowledge storage.** A node that stores a shard cannot determine the file it belongs to, its sender, its recipient, or its content. Shards are cryptographically indistinguishable from random noise.

3. **Erasure resilience.** The network tolerates the simultaneous loss of up to one-third of any file's fragments without data loss.
4. **Browser independence.** The protocol runs as a native process. It does not depend on web browsers and is therefore immune to the entire class of web-based attack vectors (XSS, CORS, third-party telemetry).

2.2 Scope

This document covers:

- Transport layer: QUIC/TLS 1.3, NAT traversal, packet framing
- DHT: Kademlia parameters, shard routing, peer maintenance
- File transfer: chunking, Reed-Solomon erasure coding, encryption, magnet links
- Cryptographic identity: proof-of-work node admission, key storage, Ed25519 signatures
- Peer naming: deterministic display names
- Local storage: shard format, quota, retention, cooperative repair
- Network resilience: rate limiting, relay, hole punching
- Command reference and protocol constants

2.3 Threat Model

Shardnet is designed to protect against the following adversaries:

Adversaries in scope:

- **Passive network observer (local/ISP).** Sees QUIC/UDP traffic between nodes. Cannot decrypt content (TLS 1.3 in transit, AES-256-GCM at rest). Can observe IP addresses, packet volumes, and timing patterns.
- **Malicious storage node.** Stores shards on behalf of the network. Sees only AES-256-GCM ciphertext indistinguishable from random data. Cannot determine file identity, origin, destination, or content.
- **Opportunistic attacker.** Attempts spam, flood, or small-scale Sybil attacks. Mitigated by Argon2id proof-of-work, token-bucket rate limiting, and TLS mutual authentication.
- **Message replay attacker.** Attempts to replay signed chat messages. Mitigated by Ed25519 signatures over nonce ‖ content with sliding-window deduplication.

Adversaries out of scope:

- **Global passive adversary (state-level surveillance).** Can correlate traffic patterns across the entire network. Shardnet does not provide anonymity against a global observer. Users requiring anonymity against such adversaries should combine Shardnet with a network-layer anonymity tool.
- **Large-scale Sybil attack (botnet).** Argon2id proof-of-work raises the cost of identity creation but does not make it prohibitive for a well-funded adversary. Shardnet is designed for community-scale deployments, not adversarial environments with nation-state resources.
- **Magnet exfiltration.** The magnet encodes the file decryption key (see §6.4). An attacker who obtains the magnet by social engineering, log exfiltration, or prompt injection can fully decrypt the file. Magnet security is the responsibility of the application layer.

Properties NOT guaranteed:

- **Anonymity of the communication graph.** DHT routing tables and STUN-visible IPs mean that "who communicates with whom" may be observable by a sufficiently positioned adversary.

- **Deniability.** Node identity is tied to a persistent Ed25519 keypair and an Argon2id proof-of-work. A node cannot plausibly deny participation in the network.
- **Forward secrecy of stored files.** AES-256-GCM keys are static per file. Compromise of a magnet retroactively compromises all past and future downloads of that file.

For the current security audit status, known limitations, and responsible disclosure process, see `SECURITY.md`.

---

3 System Architecture

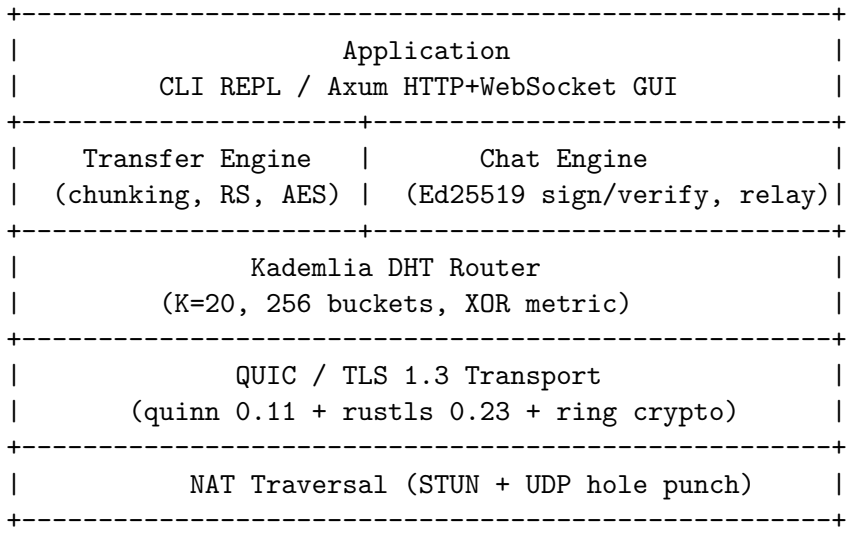
3.1 Node Model

Every Shardnet participant runs an identical binary (`shard-cli` or `shard-gui`). There is no distinction between client and server roles. Each node simultaneously:

- Stores shards on behalf of the network (up to its configured quota)
- Retrieves shards from peers on demand
- Participates in DHT routing queries
- Relays STUN and NAT punch coordination messages

A **seed node** is a well-known public node with a stable IP address (`shardnet.app:9100`). It provides the initial entry point for DHT bootstrapping but otherwise behaves identically to any other node.

3.2 Component Overview



---

4 Transport Layer

4.1 QUIC over UDP

All node-to-node communication uses QUIC (RFC 9000) over UDP. The implementation uses the `quinn 0.11` library with `rustls 0.23` and the `ring` cryptographic backend.

Parameter	Value
ALPN identifier	`shard-v1`

Parameter	Value
Protocol version	1
Max idle timeout	120 s
Keep-alive interval	10 s
Max concurrent conns	100
Max payload size	10 MB
Default P2P port	9000
Default GUI port	9201

The 120-second idle timeout is intentionally generous to accommodate Android Doze mode, which suspends network activity for extended periods.

4.2 Packet Framing

Every packet begins with an 8-byte header:

```
+----------+---------+----------+----------+
| Type (1) | Ver (1) | Rsvd (2) | Len (4)  |
+----------+---------+----------+----------+
```

Payload follows immediately. The receiver validates header bounds, rejects payloads exceeding 10 MB, and drops packets with an unknown protocol version.

Packet types:

ID	Name	Purpose
1	`Fragment`	Store a data or parity shard
2	`FragmentRequest`	Request a specific shard by hash
3	`DhtQuery`	Kademlia node lookup
4	`DhtResponse`	Kademlia lookup response
7	`ChatMessage`	Signed room broadcast
10	`StunRequest`	NAT discovery request
11	`StunResponse`	Observed public IP:port
12	`NatPunchRequest`	Initiate simultaneous open
13	`NatPunchRendezvous`	Seed relays handshake coordination
14	`KeepAlive`	Heartbeat
15	`RelayData`	Wrapped packet for multi-hop relay

4.3 TLS Authentication

Nodes use self-signed X.509 certificates generated at first launch. The certificate embeds a proof-of-work salt in its `OU` field (see §6.1). Mutual TLS authentication is enforced: both endpoints present their certificate and the peer verifies proof-of-work validity before accepting the connection.

4.4 NAT Traversal

Shardnet nodes behind NAT operate without port forwarding via two mechanisms:

STUN. Every node sends a `StunRequest` to the seed node at startup and every 60 seconds thereafter (with a 30-second initial delay). The seed replies with a `StunResponse` containing the observed public `IP:port`, which the node registers in the DHT as its reachable address.

UDP hole punching. When two nodes behind separate NATs need to connect, the seed relays a `NatPunchRendezvous` message to both. Each node sends simultaneous UDP packets to the other's public address, creating NAT mappings on both sides. The implementation retries up to 5 times at 800 ms per attempt (200 ms between attempts), constrained to avoid triggering QUIC's 10-second connection timeout.

Relay fallback. If direct connectivity cannot be established, packets are wrapped in `RelayData` packets with a TTL of 3 hops and forwarded through intermediate peers.

LAN preference. Nodes sharing a public IP address detect each other via `if-addrs` and route through the local network address, bypassing NAT entirely.

---

5 Distributed Hash Table

5.1 Kademlia Parameters

Shardnet uses a Kademlia-variant DHT with 256-bit node IDs.

Parameter	Value
K (bucket size)	20
Bucket count	256
Replacement cache per bucket	20
Ping timeout	1 s
Grace period	15 s
Eviction threshold	3 consecutive failures
Maintenance cycle	30 s (5 min in sleep mode)

Distance metric. The XOR distance between two node IDs is computed on their raw 32-byte representations. The bucket index for a given distance is the position of its most-significant set bit:

$$d = \mathrm{ID}_A \oplus \mathrm{ID}_B$$

$$\mathrm{bucket}(d) = (31 - \lfloor \log_2 d_{\mathrm{byte}} \rfloor) \cdot 8 + (7 - \mathrm{clz}(d_{\mathrm{byte}}))$$

where d_{byte} is the first non-zero byte of d and clz is count-leading-zeros.

5.2 Node Lookup

To find nodes closest to a target hash, a node queries its K closest known peers and iteratively narrows toward the target using standard Kademlia lookup. Results are sorted by XOR distance; the K closest surviving peers are returned.

5.3 Shard Routing

Each shard is addressed by a deterministic 32-byte key:

$$k_{\text{shard}} = \text{SHA-256}(\text{file\_id} \parallel \text{chunk\_index} \parallel \text{shard\_index})$$

where file_id is 32 bytes, chunk_index and shard_index are 4-byte big-endian integers. This key is used both as the DHT lookup target and as the shard's storage filename on disk (hex-encoded).

On upload, each shard is pushed to the 3 DHT-closest nodes for its key. On download, a `FragmentRequest` is sent to those same nodes.

5.4 Peer Maintenance

A background task runs every 30 seconds (every 5 minutes when no traffic is detected). It pings all known peers, marks unresponsive ones, and evicts peers that have failed 3 consecutive health checks after a 15-second grace period. Evicted slots are filled from the per-bucket replacement cache.

Bootstrap order on startup: 1. Cached peers from previous session 2. `--bootstrap` CLI argument 3. Public seed: `shardnet.app:9100` 4. Local fallback: `127.0.0.1:9100`

---

6 File Transfer Protocol

6.1 Chunking

Files are split into fixed-size chunks of **1 MB** (1,048,576 bytes). The final chunk is padded with random bytes to the nearest shard boundary. A 10-byte header is prepended to the chunk stream:

```
+------------------+-------------+------------------+
| File size (8 B)  | Name len (2B)| Filename (var.)  |
+------------------+-------------+------------------+
```

This header is encrypted along with the payload; peers storing shards have no access to the filename or file size.

6.2 Erasure Coding

Each chunk is processed by Reed-Solomon erasure coding (library: `reed-solomon-erasure 6.0`):

Parameter	Value
Data shards (k)	10
Parity shards (m)	5
Total shards (n)	15
Reconstruction threshold	≥ 10 shards
Max simultaneous node failures tolerated	5

$$n = k + m = 10 + 5 = 15$$

Any subset of $k = 10$ shards out of $n = 15$ is sufficient to reconstruct the chunk. The network tolerates the simultaneous loss of up to $m = 5$ shards — one third of all fragments.

6.3 Encryption

Before erasure coding, each chunk is encrypted independently:

- **Cipher:** AES-256-GCM (`aes-gcm 0.10`)
- **Key:** 32 bytes, randomly generated per file
- **Nonce:** 12 bytes, randomly generated per chunk
- **Scope:** The nonce is stored with the ciphertext; the key is never stored on the network

The key is embedded in the magnet link and never transmitted separately. A node storing a shard sees only opaque ciphertext; it cannot distinguish a shard from random data.

6.4 Magnet Links

A magnet link encodes everything needed to retrieve and decrypt a file:

$$\text{magnet} = \text{BASE64\_URL\_NO\_PAD}(\text{file\_id}_{32} \parallel \text{master\_key}_{32})$$

Total encoded length: 86 characters. The `file_id` is used to reconstruct shard addresses via the DHT key formula (§4.3). The `master_key` decrypts every chunk. Sharing a magnet link grants full read access to the file.

Security note. The magnet encodes the file decryption key. Confidentiality of the file depends entirely on the confidentiality of the magnet. A compromised magnet grants full read access regardless of shard distribution or network topology. Treat the magnet as a high-value secret: do not log it in plaintext, do not include it in prompts or debug traces, and do not transmit it over unencrypted channels.

6.5 Shard Distribution and Retrieval

Upload path:

1. Read file, prepend header, pad to chunk boundary
2. For each chunk: encrypt with AES-256-GCM, apply Reed-Solomon, obtain 15 shards
3. For each shard i: compute k_{shard}, find 3 closest DHT peers, send `Fragment` packet
4. Return magnet link to the user

Download path:

1. Decode magnet link: extract `file_id` and `master_key`
2. For each chunk: send `FragmentRequest` to the 3 closest peers for each of the 15 shard addresses
3. Accept the first 10 valid responses per chunk
4. Reed-Solomon reconstruct; AES-256-GCM decrypt
5. Strip header, write file to `./downloads/`

Concurrency: Up to 5 transfers run in parallel, enforced by a semaphore.

---

7 Cryptographic Security

7.1 Node Identity and Proof-of-Work

At first launch, a node generates an Ed25519 key pair and a self-signed X.509 certificate. The certificate embeds a 32-byte random salt in its `OU` field. To be admitted to the network, the node must produce a valid proof-of-work:

$$\text{Argon2id}(\text{cert\_DER} \parallel \text{salt}, \text{params}) \rightarrow h$$

The leading d bits of h must all be zero (default: $d = 6$). This requires approximately 2^d hash evaluations and takes 1–5 seconds on a general-purpose CPU.

Argon2id parameters:

Parameter	Value
Memory	65,536 KB
Iterations	1
Parallelism	1
Output size	32 bytes
Default difficulty	6 bits

The node ID is derived from the same operation:

$$\text{NodeID} = \text{Argon2id}(\text{cert\_DER} \parallel \text{salt})$$

This ties the node's network identity irrevocably to its certificate and proof-of-work. Changing either requires regenerating the node ID.

7.2 Key Storage

Private keys are stored encrypted at rest:

Item	Path
Private key (PKCS8)	`{storage}/sys/node_key.enc`
Certificate (DER)	`{storage}/sys/node_cert.der`

Encryption format:

```
[ salt (32 B) ][ nonce (12 B) ][ ciphertext ]
```

The encryption key is derived as:

$$K_{\text{enc}} = \text{SHA-256}(\text{machine\_id} \parallel \text{salt})$$

where `machine_id` is obtained from `/etc/machine-id` (Linux) or a persisted UUID (Android). The key file is unreadable on a different machine. This defends against an attacker who obtains a copy of the storage directory — from a backup, a disk image, or storage media theft — without physical access to the running host.

NTP synchronization. Each node synchronizes its clock with `pool.ntp.org` at startup and corrects for drifts exceeding ± 0.001 s, hardening against replay attacks.

Key lifecycle. The Ed25519 keypair is generated once at first launch. The node ID is derived irrevocably from the same Argon2id operation as the certificate; changing the keypair requires

generating a new node identity and permanently abandoning the old one. No rotation or revocation mechanism exists in the current protocol.

Compromise response. If a node's private key is compromised:

1. Stop the node immediately.
2. Delete `{storage}/sys/node_key.enc` and `{storage}/sys/node_cert.der`.
3. Restart — a new keypair, certificate, and node ID are generated automatically.
4. The compromised identity cannot be actively revoked in the DHT. Other nodes will evict it after 3 consecutive failed health checks (grace period: 15 s).

Scope of compromise. Ed25519 key compromise does not expose stored file content — files are encrypted with separate AES-256-GCM keys carried exclusively in magnet links. It does enable an attacker to impersonate the node and produce valid Ed25519 signatures on chat messages for the duration until eviction.

7.3 Zero-Knowledge Storage

The shard key k_{shard} is a preimage-resistant commitment to the shard's position in the file. A storage node does not know the `file_id`, the chunk or shard index, nor can it link two shards to the same file. It sees only AES-GCM ciphertext, computationally indistinguishable from random data.

7.4 Shard Integrity Verification

On retrieval, the downloader verifies that the received shard's hash matches the expected k_{shard}. Corrupted or tampered shards are rejected immediately and a replacement is requested from another peer.

A 16 MB ceiling (`MAX_RECONSTRUCTION_SIZE`) prevents memory exhaustion from maliciously crafted shard geometries.

---

8 Peer Identity and Naming

8.1 Deterministic Display Names

Each node is assigned a human-readable display name derived deterministically from its node ID via a djb2-variant hash over the first 8 hex characters:

$$h = \left(\sum_{c \in \text{id}[0..8]} h \cdot 31 + c \right) \bmod 256 \qquad \text{name} = \text{PEER\_NAMES}[h]$$

`PEER_NAMES` is a fixed table of 256 short given names. The same algorithm runs identically in the CLI and the GUI, guaranteeing a given node ID maps to the same name across all interfaces.

Users may override their display name with `/name <alias>`, broadcast as a `\x1fNAME:<alias>` system message.

8.2 Machine/Human Message Separator

Messages prefixed with ASCII Unit Separator `\x1f` (U+001F) are system messages, filtered from human-readable display:

Prefix	Meaning
`\x1fNAME:<n>`	Sender announces display name

API and WebSocket clients receive all messages including system prefixes.

8.3 Chat Message Integrity

Every chat message is Ed25519-signed by the sender:

$$\sigma = \text{Ed25519Sign}(\text{room\_name} \parallel \text{sender\_id} \parallel \text{nonce} \parallel \text{content})$$

The `nonce` is a `u64` counter preventing replay. Receivers verify the signature and check the nonce against a two-generation rolling deduplication cache before relaying.

---

9 Local Storage

9.1 Shard Storage Format

Shards are stored as raw binary files in a flat directory:

```
{storage_path}/
+-- sys/
|   +-- node_key.enc      # Encrypted Ed25519 private key
|   +-- node_cert.der     # X.509 self-signed certificate
+-- <64-char hex key>     # One file per stored shard
```

The filename is the lowercase hex encoding of k_{shard}. Filenames are validated at read and write time: exactly 64 characters, lowercase hex only — all other paths are rejected to prevent directory traversal.

9.2 Quota and Retention

Parameter	Default
Max storage	500 MB
Retention period	7 days (604,800 s)
Cleanup interval	30 min (1,800 s)

The cleanup task runs every 30 minutes. It deletes any shard whose filesystem `mtime` exceeds the retention period. The same window applies to `./downloads/`. All three parameters are live-configurable via `PATCH /config` without restart.

9.3 Cooperative Shard Repair

When a node reconstructs a file chunk from fewer than 15 shards, it regenerates and re-pushes the missing shards in a detached background task:

1. Identify missing shard indices from the download
2. Regenerate them via Reed-Solomon from the reconstructed plaintext
3. Recompute k_{shard} for each missing index
4. Push to the 3 DHT-closest nodes for each key

This repair runs without delaying the download response. Over time, it counteracts shard erosion from node churn without a dedicated archival daemon.

---

10 Network Resilience

10.1 Rate Limiting

Incoming connections are rate-limited per source IP using a token bucket:

Parameter	Standard	Relay
Bucket capacity	20 tokens	5 tokens
Refill rate	10 / s	2 / s
Cache cleanup	300 s	300 s

10.2 Relay

When peers cannot connect directly, `RelayData` packets are forwarded through intermediaries:

- **TTL:** initialized to 3, decremented per hop, dropped at 0
- **Relay selection:** 3 DHT-closest nodes to the target
- **Rate:** subject to the relay token bucket

10.3 File Descriptor Limits

At startup, the node raises its file descriptor limit to 80% of the OS maximum (minimum 50), enabling up to 100 concurrent QUIC connections alongside open shard files.

10.4 Defensive Robustness Properties

The following protections operate at the network and storage layers to resist resource exhaustion and denial-of-service conditions. They are independent of the higher-level cryptographic guarantees described in §7.

Property	Mechanism	Reference
Packet size cap	10 MB hard payload limit; packets exceeding this are dropped at the framing layer before processing	§3.1, §3.2
Reconstruction memory cap	`MAX_RECONSTRUCTION_SIZE` = 16,777,216 B; reconstruction aborts if the projected shard geometry would exceed this bound	§7.4
Per-IP rate limiting	Token-bucket per source IP (capacity 20, refill 10 s ^{1}); relay path uses a stricter independent bucket (capacity 5, refill 2 s ^{1})	§10.1

Property	Mechanism	Reference
Directory traversal prevention	Shard filenames are validated as exactly 64 lowercase hex characters at both read and write time; any non-conforming path is rejected unconditionally	§9.1
Machine-bound key isolation	The private key file is AES-GCM-encrypted with a key derived from `machine_id`; the keystore cannot be decrypted by copying it to a different physical or virtual machine	§7.2
Connection ceiling	The QUIC endpoint is hard-capped at 100 concurrent connections	§3.1

---

11 Command Reference

11.1 CLI Commands

Command	Description
`/put <file>`	Encrypt, shard, and upload a file; prints magnet
`/get <magnet>`	Download and decrypt a file to `./downloads/`
`/read <magnet>`	Fetch and render a Markdown shard in the terminal
`/join <room>`	Join a named chat room
`/leave`	Leave the current room
`/name <alias>`	Set display name; broadcasts `\x1fNAME:<alias>`
`/peers`	Show routing table size
`/status`	Node info, storage usage, and config
`/exit`	Persist state and quit

11.2 Startup Flags

Flag	Effect
`--daemon`	Unix background daemon; logs to `./logs/`
`--passive`	Listen-only; `/get` and `/read` remain available
`--seed`	Seed mode; disables outgoing bootstrap
`--bootstrap <host:port>`	Explicit bootstrap peer
`--port <n>`	P2P port (default: 9000)

Flag	Effect
`--disk-quota <bytes>`	Storage quota override at startup
`--retention <secs>`	Retention period override at startup
`--gui-port <n>`	GUI HTTP port (shard-gui only; default: 9201)
`--gui-host <addr>`	GUI bind address (default: 127.0.0.1)

11.3 Runtime Configuration API

`shard-gui` exposes a REST endpoint for live configuration changes:

```
PATCH /config
Content-Type: application/json

{
  "quota_bytes":          500000000,
  "retention_sec":        604800,
  "cleanup_interval_sec": 1800
}
```

All fields are optional. Changes persist to `config.toml`.

11.4 Agent Integration Security

Shardnet exposes a REST and WebSocket API suitable for use by autonomous agents and LLM-based systems. The following risks are specific to these contexts and must be addressed at the application layer.

Magnet links in agent contexts. A magnet link encodes the file decryption key (see §6.4). LLM orchestration frameworks log full prompts, tool call arguments, and tool outputs by default. A magnet appearing in any of these surfaces — system prompt, conversation turn, tool argument, trace span — is effectively leaked to every system that processes that context.

Risk surface	Mitigation
Prompt / conversation history	Never include magnets in prompts or message history
Framework logs (LangChain, CrewAI, etc.)	Exclude download tool from trace capture
Distributed tracing (OpenTelemetry, Datadog)	Redact magnet fields before export
Vector DB / KV context store	Store a reference ID only; resolve to magnet in a separate secret store

Recommended pattern — opaque reference ID. The agent receives a short reference ID (e.g. a UUID) that maps to the magnet inside a secret manager or KMS. The agent calls a thin proxy endpoint that resolves the ID to the magnet internally and returns the file, without the magnet ever appearing in the agent's context window.

Envelope encryption. For stricter isolation, wrap the `master_key` with an application-layer key before storing or transmitting it. The agent receives the wrapped key and a reference to the wrapping key; neither is sufficient alone to decrypt the file.

12 Appendix A — Protocol Constants

Constant	Value
`PROTOCOL_VERSION`	1
K (DHT bucket size)	20
DHT bucket count	256
Chunk size	1,048,576 B (1 MB)
RS data shards	10
RS parity shards	5
RS total shards	15
Shard replication factor	3
Max concurrent transfers	5
Max reconstruction size	16,777,216 B (16 MB)
Max packet payload	10,485,760 B (10 MB)
QUIC idle timeout	120 s
QUIC keep-alive	10 s
STUN refresh interval	60 s
NAT punch attempts	5
NAT punch timeout	800 ms / attempt
Relay TTL	3 hops
Peer grace period	15 s
Peer eviction threshold	3 failures
Maintenance cycle	30 s
Default P2P port	9000
Default GUI port	9201
Default seed	`shardnet.app:9100`
PoW default difficulty	6 bits
Argon2id memory	65,536 KB
Argon2id iterations	1
Rate limit capacity	20 tokens
Rate limit refill	10 tokens / s
Relay rate capacity	5 tokens
Relay rate refill	2 tokens / s

13 Appendix B — Dependency Versions

Crate	Version	Role
`quinn`	0.11	QUIC transport

Crate	Version	Role
`rustls`	0.23	TLS 1.3 (ring backend)
`rcgen`	0.13	X.509 certificate generation
`reed-solomon-erasure`	6.0	RS erasure coding (10+5)
`aes-gcm`	0.10	AES-256-GCM symmetric encryption
`ed25519-dalek`	2.1	Ed25519 signatures and key management
`x25519-dalek`	2.0	X25519 key exchange
`argon2`	0.5	Argon2id PoW and node ID derivation
`sha2`	0.10	SHA-256 shard addressing
`axum`	0.7	HTTP/WebSocket GUI server
`bincode`	1.3	Packet serialization
`tokio`	1.32	Async runtime
`machine-uid`	0.5	Machine-bound key derivation (Linux)
`rsntp`	4.0	NTP clock synchronization
`if-addrs`	0.12	LAN address detection
`base64`	0.22	Encoding
`clap`	4.4	CLI argument parsing
`tracing`	0.1	Structured logging