

# Boing Network — Operational Runbook

**Purpose:** Operations guide for running and maintaining Boing Network nodes.

**References:** [BOING-NETWORK-ESSENTIALS.md](#) (six pillars and network essentials), [BUILD-ROADMAP.md](#), [README.md](#), [RPC-API-SPEC.md](#), [READINESS.md](#) (beta checklist and quick starts)

---

## Network essentials (six pillars)

The network prioritizes, in order: **1. Security** → **2. Scalability** → **3. Decentralization** → **4. Authenticity** → **5.**

**Transparency** → **6. True quality assurance**. For the full description of each pillar and design philosophy, see [BOING-NETWORK-ESSENTIALS.md](#).

---

## Table of Contents

- [Node Setup](#)
  - [Running a Node](#)
  - [RPC Endpoints](#)
  - [CLI Usage](#)
  - [Monitoring & Health](#)
  - [Incident Response](#)
  - [Troubleshooting](#)
  - [Testnet Operations](#)
- 

## 1. Node Setup

### Prerequisites

- Rust:** 1.70+ ( `rustup` recommended)
- OS:** Linux, macOS, or Windows (WSL recommended on Windows)

### Build

```
cargo build --release
```

### Directory Layout

Path	Description
target/release/boing-node	Node binary
target/release/boing	CLI binary
~/.boing/ or ./data/	Data directory (when using <code>--data-dir</code> )

---

## 2. Running a Node

### Full Node (non-validator)

```
cargo run -p boing-node
```

Defaults: RPC on `http://127.0.0.1:8545` .

## Validator Node

```
cargo run -p boing-node -- --validator --rpc-port 8545
```

Produces blocks when there are pending transactions.

## With Data Directory

```
cargo run -p boing-node -- --data-dir ./boing-data --rpc-port 8545
```

## Mempool: pending transactions per sender

Each account may have only **N** distinct pending nonces in the mempool at once (default **N = 16**, aligned with [RateLimitConfig::default\\_mainnet](#) ). The process logs `Mempool: max pending txs per sender = ...` at startup. Operators can raise the cap (e.g. busy testnet RPC) with:

```
cargo run -p boing-node -- --validator --rpc-port 8545 --pending-txs-per-sender 64
```

Nonce **replacements** for the same nonce do not count toward the cap.

## Dev rate-limit profile (local / busy testnet)

For **relaxed** JSON-RPC rate limits and a **64** pending-tx cap per sender (see [RateLimitConfig::default\\_devnet](#) ):

```
cargo run -p boing-node -- --validator --rpc-port 8545 --dev-rate-limits
```

Same effect without the flag (e.g. Docker / systemd):

```
export BOING_RATE_PROFILE=dev
cargo run -p boing-node -- --validator --rpc-port 8545
```

Use `BOING_RATE_PROFILE=mainnet` to force the strict profile even if `--dev-rate-limits` is mistakenly present on a public RPC host.

You can still override only the mempool cap: `--dev-rate-limits --pending-txs-per-sender 128` . **Do not** use the dev profile on public mainnet RPC endpoints.

## JSON-RPC surface (stale binary or proxy)

If `boing_chainHeight` works but other `boing_*` calls return `-32601 Method not found`, the process on the RPC port is almost certainly an **older boing-node build** or a **proxy that drops unknown methods**. Rebuild from the current repository ( `cargo build -p boing-node --release` ) and restart. To see which read-only methods the URL exposes, run `npm run build` then `npm run probe-rpc` in `boing-sdk` , or `npm run probe-rpc` from the repository root (prints a `diagnosis` when methods are missing). **No SDK build:** from the repo root run `npm run`

`rpc-endpoint-check` for a raw JSON-RPC table. **Windows — who owns port 8545:** `netstat -ano | findstr :8545`, then `tasklist /FI "PID eq <pid>" /V` (ensure the image is the `boing-node` you just built, not an old cargo install path). The tutorial package also exposes `npm run probe-rpc`. See [RPC-API-SPEC.md](#) § Diagnosing -32601.

**Public RPC — chain id for dApps:** Set environment `BOING_CHAIN_ID=6913` and `BOING_CHAIN_NAME=Boing Testnet` on the `boing-node` process so `boing_getNetworkInfo` returns them (matches wallet `boing_chainId 0x1b01`). Copy-paste template: [tools/boing-node-public-testnet.env.example](#). Wallets can still use a fixed chain id from app config when the field is `null`.

### 3. RPC Endpoints

Method	Params	Description
<code>boing_submitTransaction</code>	<code>[hex_signed_tx]</code>	Submit a signed transaction
<code>boing_chainHeight</code>	<code>[]</code>	Current chain height
<code>boing_getSyncState</code>	<code>[]</code>	Committed tip: head / finalized / latest block hash
<code>boing_getNetworkInfo</code>	<code>[]</code>	dApp discovery: tip + timing + optional <code>chain_id</code> / <code>chain_name</code> (env); explicit non-availability of chain-wide TVL / APY — <a href="#">RPC-API-SPEC.md</a>
<code>boing_getBalance</code>	<code>[hex_account_id]</code>	Spendable balance (decimal string)
<code>boing_getAccount</code>	<code>[hex_account_id]</code>	Balance, nonce, stake (for wallets and tx building)
<code>boing_getBlockByHeight</code>	<code>[height]</code>	Block at height (u64)
<code>boing_getBlockByHash</code>	<code>[hex_block_hash]</code>	Block by hash (32 bytes hex)
<code>boing_getAccountProof</code>	<code>[hex_account_id]</code>	Merkle proof for account
<code>boing_verifyAccountProof</code>	<code>[hex_proof, hex_state_root]</code>	Verify Merkle proof
<code>boing_simulateTransaction</code>	<code>[hex_signed_tx]</code>	Simulate tx (gas, success)
<code>boing_registerDappMetrics</code>	<code>[hex_contract, hex_owner]</code>	Register dApp for incentives
<code>boing_submitIntent</code>	<code>[hex_signed_intent]</code>	Submit signed intent for solver fulfillment
<code>boing_faucetRequest</code>	<code>[hex_account_id]</code>	Testnet only: request testnet BOING (node must be started with <code>--faucet-enable</code> )
<code>boing_getLogs</code>	<code>[{ fromBlock, toBlock, address?, topics? }]</code>	Bounded log query (max <b>128</b> inclusive heights, <b>2048</b> rows per call); see <a href="#">RPC-API-SPEC.md</a>

#### Public RPC operators and `boing_getLogs`

- `boing_getLogs` is more expensive than single-block reads (each call scans up to **128** committed heights and may return up to **2048** log rows). On **shared** or **rate-limited** endpoints, operators should **document** whether the method is enabled and any **extra limits** (e.g. lower QPS than `boing_chainHeight`).

- If you need to protect the node, prefer **global RPC rate limits** (see `RateLimitConfig` / node defaults) or a **reverse proxy** rule; disabling the method is a last resort and breaks indexers that rely on it—see [INDEXER-RECEIPT-AND-LOG-INGESTION.md](#).
- **Indexer clients:** use `boing-sdk getLogsChunked` (or equivalent chunking) so each RPC stays within the **128-block** cap; default `maxConcurrent: 1` on wide backfills is polite on public tiers. For full block + receipt replay, `fetchBlocksWithReceiptsForHeightRange` and `planIndexerCatchUp` are documented in the same indexer guide and in [examples/native-boing-tutorial](#) ( `npm run indexer-ingest-tick` ).
- **Internet-facing smoke (no keys):** from the tutorial package, `npm run preflight-rpc` with public `BOING_RPC_URL` exercises `check-testnet-rpc` plus a one-shot `getSyncState` sample; command matrix: [PRE-VIBEMINER-NODE-COMMANDS.md](#).

Example (curl):

```
curl -X POST http://127.0.0.1:8545/ -H "Content-Type: application/json" \
  -d '{"jsonrpc": "2.0", "id": 1, "method": "boing_chainHeight", "params": []}'
```

## 4. CLI Usage

Command	Description
<code>boing init [name]</code>	Scaffold a new dApp project
<code>boing dev [--port 8545]</code>	Start local dev chain
<code>boing deploy [path]</code>	Deploy contract or config
<code>boing metrics register --contract &lt;hex&gt; --owner &lt;hex&gt;</code>	Register contract for dApp incentives
<code>boing completions &lt;shell&gt;</code>	Generate shell completion (bash, zsh, fish, powershell, elvish)

## Shell Completion

```
# Bash
boing completions bash > /etc/bash_completion.d/boing # or ~/.local/share/bash-completion/completions/boing

# Zsh
boing completions zsh > ~/.zsh/completions/_boing

# Fish
boing completions fish > ~/.config/fish/completions/boing.fish
```

## 5. Monitoring & Health

### Chain Height

```
curl -s -X POST http://127.0.0.1:8545/ -H "Content-Type: application/json" \
-d '{"jsonrpc":"2.0","id":1,"method":"boing_chainHeight","params":[]}' | jq
```

## Block Query

```
curl -s -X POST http://127.0.0.1:8545/ -H "Content-Type: application/json" \
-d '{"jsonrpc":"2.0","id":1,"method":"boing_getBlockByHeight","params":[0]}' | jq
```

## Logs

Set `RUST_LOG` before running:

```
RUST_LOG=info cargo run -p boing-node
# Debug: RUST_LOG=debug
# Trace: RUST_LOG=trace
```

## 6. Incident Response

For security incidents and vulnerabilities:

Step	Action
1. <b>Detect</b>	Monitor logs, alerts, community reports.
2. <b>Assess</b>	Classify severity: Low, Medium, High, Critical.
3. <b>Contain</b>	Isolate affected systems; pause if necessary.
4. <b>Communicate</b>	Notify validators, users, ecosystem per severity.
5. <b>Remediate</b>	Apply fixes; coordinate upgrades via governance if needed.
6. <b>Post-mortem</b>	Document cause, impact, and prevention.

**Contacts:** [SECURITY-STANDARDS.md](#) § 5. **Security Contacts** (responsible disclosure via GitHub Security Advisories; bug bounty TBD).

## 6b. Scalability Characteristics

- **Block time:** ~2 seconds (configurable via tokenomics)
- **Throughput:** Parallel transfer batches; access-list batching reduces conflicts
- **Gas:** Fixed per tx type (Transfer, Bond, Unbond, ContractCall, ContractDeploy)
- **Batching:** Scheduler groups non-conflicting txs; transfers with disjoint access lists run in parallel

## 6c. Decentralization Design

- **Permissionless validation:** No whitelist; anyone with stake can validate
- **P2P discovery:** Bootnodes for bootstrap; mDNS for LAN; DHT (roadmap) for discovery
- **No central gatekeeper:** Consensus, governance, and QA pool are decentralized

- **Single-client today:** Multiple implementations encouraged for resilience

## 7. Troubleshooting

### Node won't start

1. Ensure port 8545 (or `--rpc-port`) is free.
2. Check `RUST_LOG=debug` for errors.
3. On Windows: ensure no firewall blocking; try WSL if TCP binding fails.

### Transaction not included

- Validator mode must be enabled for block production.
- Check mempool size and nonce ordering.
- Simulate first: `boing_simulateTransaction` to validate.
- **Note:** If block production or consensus fails, transactions are re-inserted into the mempool automatically so they can be retried in the next round.

### RPC returns "Method not found"

- Ensure you're using the exact method name (case-sensitive).
- Params must be a JSON array (e.g. `"params": []` not `"params": {}`).

### Build fails

```
cargo clean
cargo build
```

## 8. Testnet Operations

When running the **public incentivized testnet**, the following operations keep bootnodes and the faucet available. See [TESTNET.md](#) Part 3 for the full launch checklist.

### 8.1 Running a bootnode

A **bootnode** is a node with a stable, publicly reachable address that other nodes use to join the network.

1. **Build:** `cargo build --release`
2. **Run with P2P and a fixed port:**

```
./target/release/boing-node \
  --p2p-listen /ip4/0.0.0.0/tcp/4001 \
  --validator \
  --rpc-port 8545 \
  --data-dir ./bootnode-data
```

3. **Publish the multiaddr:** Your bootnode address is `/ip4/<YOUR_PUBLIC_IP>/tcp/4001`. Ensure TCP port 4001 (and 8545 if RPC is public) is open in the firewall. Add this multiaddr to [TESTNET.md](#) §6 and to `website/src/config/testnet.ts` (or set `PUBLIC_BOOTNODES` at build time).
4. **Recommendation:** Run at least **two** bootnodes on different hosts for redundancy.
5. **P2P transaction gossip:** After a successful `boing_submitTransaction` or `boing_faucetRequest`, the node gossips the **signed** transaction on `boing/transactions`. Peers verify the signature and apply the

same mempool + QA rules as RPC submissions (duplicates and rejects are dropped at debug log level).

**Regression:** `cargo test -p boing-node --test p2p_tx_gossip_rpc` (four-node full mesh: RPC submit on one peer → another peer's mempool contains the same tx id). **Note:** libp2p gossipsub's default mesh size means **two peers alone** may not propagate topic messages reliably; production testnets should run enough connected peers (or tune gossipsub — see `boing-p2p`).

6. **Connections per IP:** The active rate-limit profile caps simultaneous P2P connections from the same remote IPv4/IPv6 address (mainnet profile **50**, dev profile **100**). Override with `--max-connections-per-ip N`; use `0` for unlimited (not recommended on public listeners).

## 8.2 Running the faucet node

The **faucet** is an RPC method (`boing_faucetRequest`) on a node started with `--faucet-enable`. Use a dedicated node (or a node behind your public RPC) so the website faucet page can target it.

1. **Build:** `cargo build --release`
2. **Run with faucet enabled:**

```
./target/release/boing-node \  
  --validator \  
  --faucet-enable \  
  --rpc-port 8545 \  
  --data-dir ./faucet-data
```

For the **public testnet**, also use `--p2p-listen` and `--bootnodes` so this node syncs with the network.

3. **Publish the RPC URL:** Point users and the website to this node's RPC (e.g. `https://testnet-rpc.boing.network/`). Set `PUBLIC_TESTNET_RPC_URL` when building the website so the faucet page defaults to this URL.
4. **Chain metadata on public testnet:** Export `BOING_CHAIN_ID=6913` and `BOING_CHAIN_NAME=Boing Testnet` before starting `boing-node` (or use `Environment=` in systemd) so `boing_getNetworkInfo` exposes them — [tools/boing-node-public-testnet.env.example](#).
5. **Rate limit:** The faucet allows 1 request per 60 seconds per account; no extra config needed.
6. **Genesis:** The faucet account is funded at genesis with 10,000,000 testnet BOING (see `boing-node/src/faucet.rs`). Ensure all testnet nodes use the same genesis so the faucet balance exists.

## 8.3 Cloudflare Tunnel (testnet-rpc.boing.network)

For full setup steps, see [INFRASTRUCTURE-SETUP.md](#). If you use Cloudflare Tunnel to expose the RPC at `https://testnet-rpc.boing.network/`:

- **"Failed to initialize DNS local resolver"** — This cloudflared log message is usually harmless. The tunnel has already registered; traffic forwarding works. It occurs when cloudflared cannot reach `region1.v2.argotunnel.com` for optional region/metrics. You can ignore it. If it bothers you, try a different DNS (e.g. 1.1.1.1) or firewall rules that allow outbound DNS.
- **HTTP 530 / body error code: 1033 (or similar) from https://testnet-rpc.boing.network/** — Cloudflare is up, but the **tunnel origin** (the machine running `cloudflared` + `boing-node` RPC) is **unreachable** or the tunnel process is down. This is **not** a JSON-RPC or SDK bug: no `boing_qaCheck` / `boing_chainHeight` will succeed until ops restores the tunnel or you point clients at a **working** RPC (e.g. local `http://127.0.0.1:8545`). The **boing-sdk** surfaces the response body in the error message when the status is non-2xx.

## 8.4 Monitoring the testnet

- **Upgrading the RPC binary:** [PUBLIC-RPC-NODE-UPGRADE-CHECKLIST.md](#) (build, test, tunnel, post-deploy `check-testnet-rpc` ).
- **Chain height:** Call `boing_chainHeight` on the public RPC periodically; alert if growth stalls. Tutorial `BOING_POLL_ONCE=1 node scripts/observer-chain-tip-poll.mjs` emits one JSON sample (see [TESTNET-OPS-RUNBOOK.md](#) §3).
- **Faucet balance:** Check the faucet account balance via `boing_getBalance` with the faucet account ID; refill or alert when low (genesis funding is 10M; 1,000 per request).
- **Bootnode reachability:** Ensure ports 4001 (P2P) and 8545 (RPC, if exposed) are reachable from the internet; use a simple TCP check or your monitoring stack.

### 8.5 Log: P2P: block publish error: InsufficientPeers (Gossipsub)

**What it means:** libp2p Gossipsub only forwards a published message to peers that have **advertised subscription** to the topic ( `boing/blocks` / `boing/transactions` ). Right after startup—or on very small networks—there can be a **short window** where you are connected (or still dialing) but **no peer is in `topic_peers` yet**, so `publish` returns `InsufficientPeers` .

**What is *not* broken:** Local consensus already **committed** the block (you will see `Consensus: committed block` in the same trace). JSON-RPC, faucet, and wallet balance use the node’s local chain, not Gossipsub.

**Propagation:** Other nodes can still obtain blocks via the **block request/response** protocol ( `/boing/block-sync/1` ) once they are connected; Gossipsub is an optimization for fan-out, not the only sync path.

**If the second node never catches up:** Check bootnode multiaddrs, firewall **TCP 4001**, matching genesis, and that both sides run a build with the same network ID / chain config—not this warning alone.

### 8.6 “Smart contracts”, boing.finance, and Boing devnet

**Boing L1 uses the Boing VM only.** Execution is the stack machine + opcodes in `crates/boing-execution` , with bytecode QA in `boing-qa` . Contracts are deployed with on-chain `ContractDeploy` / called with `ContractCall` payloads inside Boing `Transaction` s, submitted as `boing_submitTransaction` (see `docs/RPC-API-SPEC.md` ). **boing.finance** must not treat **chain 6913** as an EVM chain: its `contracts.js` entry uses **32-byte** `AccountId` fields ( `nativeConstantProductPool` , optional `nativeVm.*` module ids from build env), not Solidity factory/router addresses.

**To get on-chain programs on devnet:**

1. **Author bytecode** accepted by the protocol QA gate (see `docs/QUALITY-ASSURANCE-NETWORK.md` and `mempool/RPC QA checks`).
2. **Build and sign** a `ContractDeploy` transaction with the Boing signing model (BLAKE3 + Ed25519 + bincode), or use tooling that outputs `boing_submitTransaction -compatible hex` (CLI/SDK as they mature).
3. **Run nodes** with the same genesis and peer connectivity so blocks (and deploy txs) propagate.

**To extend boing.finance Swap / pools / lockers on Boing:** implement **Boing RPC + `ContractCall`** (and Express signing) against operator-published VM contracts; wire their **AccountIds** into `env / contracts.js nativeVm` . EVM-only code paths in that app intentionally skip **chain 6913**.