Bopwire — Technical White Paper

A peer-to-peer music network where the listeners are the infrastructure, content is verified by cryptographic fingerprint, and every play settles on a purpose-built blockchain.

How to read this paper

Each section is written in two layers. The **plain-language** paragraphs explain *what* a piece does and *why* it exists — no prior knowledge assumed. The **Under the hood** notes give the exact mechanism, sizes, and algorithms so an engineer can implement or audit it. Diagrams are ASCII so they render anywhere.

> This is a technical description of how the system works. It is not financial advice and makes no claim about the monetary value of any token; the token is described purely as the network's internal accounting and reward unit.

Contents

15. Decentralization posture

---

1. The problem

Streaming services are three businesses stacked together: a **catalogue** (rights deals), a **content-delivery network** (servers that ship the audio), and a **payment rail** (who gets paid, how much, when). The artist sits at the bottom of that stack and receives a few tenths of a cent per stream, opaquely, weeks later. The CDN is a fixed cost the platform pays for and controls.

Bopwire re-arranges that stack so the **people listening also supply the delivery network**, the **payment is settled per play on a public ledger**, and no single company owns the pipe. The hard part isn't any one of those ideas — it's making them work *together* without a central server in the middle.

---

2. The core idea

Three mechanisms are wired into one loop:

1. **Listeners are the CDN.** When you have a song, your device *seeds* it to

other listeners — like BitTorrent. Bandwidth scales with the audience instead of with a server bill.

1. **Everyone who did work gets paid per play, on-chain.** Not just the artist —

the **peer that uploaded the bytes** (the *seeder*) and the **relay that carried them** are each credited a token every time a song is genuinely listened to.

1. **Content is identified by what it sounds like.** An audio *fingerprint*

collapses every upload of the same song into one canonical entry, so the swarm pools all copies and the royalty always lands on one artist — regardless of who uploaded which file.

The result is a self-bootstrapping delivery network where the users *are* the infrastructure and are compensated for it, settled on a ledger no one privately controls.

---

3. Network topology (who talks to whom)

Four roles exist. A single device can play several of them at once (a phone is a listener, a seeder, and a wallet simultaneously).

In plain terms. Listeners (players) don't usually connect to each other directly — they connect to **mini-nodes**, which are relays sitting on reachable public IP addresses. The mini-node passes audio between two listeners (one seeding, one downloading). The **full node** is the bookkeeper and rule-keeper: it knows who holds what, runs the rules that decide a "real" play, and writes the blockchain.

Under the hood.

- Transport is **librats**, a custom C++ peer-to-peer library (QUIC/TCP messaging +

a binary side-channel). Ports: full node `p2p_port 9333`, rats RPC `9080`, a JSON-RPC API on `8545`; mini-node rats port `8080`.

- Identity is unified: a peer's **rats peer-id == its wallet address** (20-byte

secp256k1 address). One key is your network identity *and* your wallet.

- Mini-nodes exist because most listeners are behind NAT and can't accept inbound

connections; the relay is the rendezvous. It's also a deliberate economic choice (see §8).

---

4. Anatomy of a play (how data propagates, end-to-end)

This is the central data-flow. Follow a single song from "tap play" to "everyone paid."

Step by step.

- ① **Discover.** The listener asks the full node *who has this content_hash*. The

node answers with a list of online peers that hold it **plus the song's piece manifest** (the list of per-piece checksums — see §6). One round-trip; the manifest means the listener already knows the file's exact size and shape.

- ② **Fetch.** The listener requests a *range of pieces* with `swarm.fetch`. The

request travels to a seeder **through a mini-node** (`relay.forward`); the seeder streams the bytes back as **binary frames** through the same relay. The seeder paces itself to ≤4 Mbit/s per stream. The listener verifies each 256 KB piece against the manifest's SHA-256 as it arrives; a corrupt or lying piece fails its hash and is refetched from a different seeder.

- ③ **Account.** Independently, the player opens a **session** with the full node,

sends a position heartbeat every 5 seconds while actually playing, and on finish reports `session.complete` — including the **addresses of the seeder and relay** that served this play.

- ④ **Settle.** The node checks the play is genuine (§9), signs a *PlayProof*, and

mints the five reward lanes into the next block. Balances update.

Why two parallel tracks? The **audio** path (①②) and the **accounting** path (③④) are deliberately decoupled. Audio flows peer-to-peer through relays; payment flows through the full node's session pipeline. They meet only at the end, when the session reports who served the bytes.

---

5. Content identity — fingerprinting & canonical hashes

In plain terms. Two people can upload "the same song" as different files (different bitrate, different encoder, a re-rip). Bopwire recognizes them as the *same song* by listening to them, not by comparing file bytes. That way the swarm pools every copy together and the artist earns no matter which copy played.

Under the hood.

- Every song carries two hashes: a `content_hash` (SHA-256 of the exact audio

bytes — addresses *this file*) and a `fingerprint_hash` (SHA-256 of a compressed **Chromaprint** acoustic fingerprint — addresses *this recording*).

- On ingest the player computes both and submits via `fingerprint.submit` . The full

node matches against the chain three ways: exact `fingerprint_hash` , exact `content_hash` , then a **fuzzy Chromaprint similarity** search (bucketed index, similarity ≥ threshold) so different encodings of one recording **canonicalize** to a single on-chain song entry.

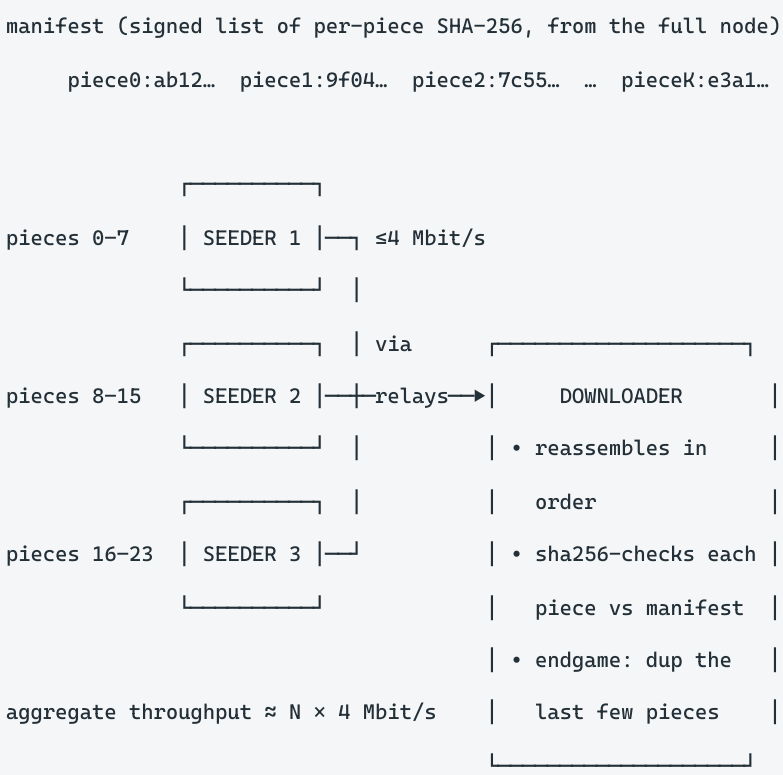
- The canonical entry lives in a block's `SongSection` (artist address, royalty

splits, metadata, duration). Variants register against it instead of duplicating.

6. The transfer layer — multi-source swarm with verified pieces

This is "Swarm Transfer v2," the layer that actually moves the audio.

In plain terms. A song is cut into fixed 256 KB pieces. A download can pull different pieces from *several seeders at once*, so your speed is the sum of their upload speeds, not any one peer's. Every piece arrives with a checksum the uploader *can't fake*, so pulling from strangers is safe — a bad piece is detected instantly and refetched elsewhere.



Under the hood.

- **Pieces:** 256 KB. **Ranges:** a `swarm.fetch{piece_start,count}` pulls up to 16

pieces (4 MB) per request, so one round-trip amortizes over many pieces.

- **Manifest:** generated at ingest — `sha256` of each piece — and served by the

full node alongside the peer list. Per-piece verification is what makes *untrusted* multi-source safe: a bad piece fails its own hash (you know *which* one) instead of only discovering corruption after assembling the whole file. The whole-file `content_hash` is the final backstop.

- **Wire format:** binary, not base64 — pieces stream as `F`-frames

`[stream_id | seq | eof | payload]` through the relay's binary channel (~33% less overhead than a JSON/base64 reply).

- **Flow control:** each seeder paces its own upload to a **4 Mbit/s** per-stream

cap; a downloader opens **one flow per seeder** (a process-global per-seeder window) and aggregates across many — so total speed scales with swarm size while no single seeder is overloaded. Streaming biases to the next-needed pieces from the fastest seeders; bulk download fans wide. An **endgame** mode duplicate-fetches the final pieces so one slow seeder can't stall completion.

---

7. The discovery layer — a private DHT

In plain terms. Beyond asking the full node "who has this song," peers also keep a distributed phone-book (a DHT) so discovery survives even if a node is busy. The important property: this phone-book is **private** — it only talks to other Bopwire nodes, never the public BitTorrent network.

Under the hood.

- The DHT speaks the standard Kademlia/KRPC wire format **but every packet carries a

private network tag** (a top-level `"mc":"mcnet1"` key). Any inbound packet missing the tag is dropped at decode — so the public BitTorrent mainline (all the `:6881` nodes) can never enter the routing table, and Bopwire nodes never answer them. It bootstraps only from configured Bopwire nodes, never the public routers (`router.bittorrent.com`, etc.).

- This keeps the routing table full of *only* Bopwire peers, so content lookups

aren't diluted by millions of unrelated nodes. Bump the magic to fork a separate network (e.g. testnet vs mainnet) — nodes with mismatched tags simply ignore each other.

- In practice the **full node's SwarmIndex** is the primary, fast discovery path

("who's online and holds hash X"); the DHT is the resilient fallback.

---

8. The relay layer — NAT traversal as a paid role

In plain terms. Most phones and home machines can't accept incoming connections (they're behind NAT/firewalls). The **mini-node** is a relay on a reachable address that bridges two such peers. Crucially, relaying is a **paid job** — a mini-node earns a token every time it carries a stream — so there's an incentive to run the infrastructure that keeps the network reachable.

Under the hood.

- A seeder's bytes flow `seeder → mini-node → downloader`. The relay forwards opaque

`F`-frames (`relay.forward` for control, `relay-bin` for audio) and is content-agnostic — it doesn't parse the audio, it just bridges.

- The relay is also a deliberate economic anchor: routing the data path through a

paid relay is what funds the reachable-IP layer. (Direct peer connections + ICE hole-punching exist in the stack but are off by default — they proved flaky and they'd bypass the relay reward.)

- Backpressure lives at the seeder (the 4 Mbit/s pace); the relay enforces only

anti-flood limits. There is no per-destination throttle — a download can saturate a relay, and you scale by adding mini-nodes, not by throttling tenants.

---

9. The chain — blocks, proofs, and the play lifecycle

In plain terms. The blockchain is the shared, tamper-evident record of *what was played and who got paid*. A play only becomes money if it passes anti-cheat rules — you can't loop a 5-second clip or fake 10,000 plays to print tokens.

The session lifecycle. A play is a three-message conversation with the full node:

```
session.start      → node opens a session (content_hash, listener)
session.heartbeat  → player reports playback position every ~5s
      … (repeat while actually playing) …
session.complete   → player says "done" (+ seeder & relay addresses)
                     node runs the gates, then mints if it passes
```

The anti-cheat gates (run at `session.complete`).

- **Coverage gate:** the union of *distinct* position ranges actually heard must

cover **≥50%** of the song's registered duration (falls back to ≥30s if duration is unknown). Re-listening the same 5 seconds collapses to 5 seconds of coverage — you can't loop a clip to qualify.

- **Density gate:** heartbeats must arrive at a plausible rate (≥1 per ~10s of wall

time), so you can't fast-forward and "complete" instantly.

- **Replay protection:** each `session_id` is single-use, persisted on-chain; a

reused session is rejected.

The proof. A passing play produces a signed `PlayProof` — session id, content hash, block hash, artist, listener, serving-node id, timestamps, duration, heartbeat count, and (v2) the **seeder** and **mini-node** addresses — signed by the full node's key. The node embeds it in a `MintTx` with the computed reward outputs and applies it to the next block.

Under the hood.

- Transactions: `TRANSFER`, `MINT` (the play reward), `RELAY_REWARD` (legacy),

`MODERATOR_OP`, `MODERATOR_PROPOSAL`, `USERNAME_REGISTER`, `SLASH`. State is LevelDB; pending txs sit in a mempool and a candidate-block builder drains them.

- `PlayProof` is **versioned by length** for forward-compatibility: legacy proofs

deserialize without the seeder/mini fields and keep validating, so adding the new reward lanes required **no chain reset**.

- The chain is currently produced/validated by a single full node (a permissioned

validator); light-client players don't validate blocks, they trust the node's signed results. Multi-validator consensus is a forward step (see §15).

---

10. Economics — the five reward lanes, supply, and burn

In plain terms. Every genuine play mints tokens to **everyone who made it happen** — not just the artist. The token is the network's internal unit of account and reward; it is consumed and earned by participation.

```
ONE qualifying play  ──────►  mint 1 token to each lane:


  ┌──────────────────────────────────────────────────────────┐
  │  ARTIST      → escrow_address(artist)  (released on KYC)  │
  │  FULL NODE   → the validator that processed the play      │
  │  DISCOVERER  → the listener            (pre-threshold tier) │
  │  SEEDER      → the peer that served the bytes             │
  │              (skipped if seeder == listener — no self-seed) │
  │  MINI-NODE   → the relay that carried the stream          │
  └──────────────────────────────────────────────────────────┘


After a song passes 10,000 plays:
  • artist + node + seeder + mini-node lanes continue
  • the discoverer lane is replaced by a listener BURN that
    grows cubically as total supply approaches the cap
```

Under the hood.

- Precision: `1 token = 1e8` internal units (like satoshis). Each lane is exactly

`1.00000000`.

- **Royalty splits:** the artist lane is divided by on-chain basis-points among

collaborators/labels (validated to sum ≤ 100%; any remainder routes to an unclaimed escrow).

- **Supply control.** `SUPPLY_FLOOR = 1,000,000,000` tokens and `SUPPLY_CAP =

2,000,000,000`. Below the floor there is **no burn** ("give the network away" bootstrap). Between floor and cap a per-play **listener burn** scales *cubically* — roughly: 1.0 B supply ⇒ 0 burned, 1.5 B ⇒ ~125, 1.8 B ⇒ ~512 — so usage gets deflationary pressure as the chain approaches the cap, and minting refuses past it.

- **The mini-node lane is per-stream, not per-byte.** Earlier the relay earned a

byte-metered `RelayRewardTx`; that is retired in favor of a flat 1-token-per-play credit. (The old transaction type is kept only so historical blocks still replay.)

---

11. Wallets, transfers, and non-custodial escrow

In plain terms. Your wallet is a key on your device. Sending tokens is a message *you* sign — the network never moves your money without your signature, and the nodes never hold your private key. Artists' pending earnings sit in **escrow held by the blockchain itself**, not by any operator's wallet.

Under the hood.

- **Transfers** are `TransferTx{from, to, amount, nonce, from_pubkey, signature}`.

The signer covers `chain_id | from | to | amount | nonce | from_pubkey` with secp256k1; the chain verifies the signature, cross-checks `address_from_pubkey == from`, enforces a per-sender **nonce** (replay protection), and checks balance. The private key never leaves the device; a node only *validates and relays* a signed transaction (the `eth_sendRawTransaction` model). `MC_CHAIN_ID = 19779` is mixed into the signature so it can't replay on another chain.

- **Escrow is non-custodial.** A pre-threshold artist reward is minted to a

deterministic `escrow_address(artist)` — funds the *chain* holds, not an operator. Release is a `RELEASE_ESCROW` proposal approved by a **moderator quorum** (majority vote), and it can only pay out **to the artist the escrow was created for**. No single party custodies the funds; a signer **cannot divert them to themselves**. The trusted part is *identity verification* (is this really the artist), not *custody* — a narrower role than holding a pot of money.

12. Security model

Surface	Mechanism
Value transfer	secp256k1 signature + inline pubkey cross-checked to `from` address + per-sender nonce (anti-replay).
Play minting	Node-signed `PlayProof`; single-use `session_id`; 50% coverage + heartbeat-density gates.
Piece integrity	Per-256 KB SHA-256 manifest verified on arrival; whole-file `content_hash` backstop.
Network isolation	Private DHT tag drops all non-Bopwire packets; private bootstrap only.
Escrow	Funds chain-held at a deterministic address; destination-bound; quorum-released; not operator-custodied.
Sybil / cheating	Permissioned, vouched moderators; cubic burn past the play threshold; replay-proof session/nonce sets.
Identity	One secp256k1 key = peer-id = wallet address (single identity across network and ledger).

13. Governance & moderation

In plain terms. A small, accountable set of moderators handles the things a public vote *can't* safely do — verifying an artist's identity to release escrow, and acting on takedown (DMCA) requests. New moderators must be *vouched for*, because a flood of fake moderators approving fake identity claims would be an attack.

Under the hood.

- A three-tier role model on-chain: `FOUNDER` (bootstrap authority, can grant/revoke

moderators), `OP` (proposes and votes on actions), `VOICE` (observer). Multi-step actions (hide content, release escrow) go through `ProposalTx` and execute on **OP-majority quorum** in-block — no single

operator acts alone.

- **Takedown without cutting the artist:** a "hidden" track is delisted from

discovery but its on-chain royalty entry persists — so a DMCA-complied song can still settle the artist's earnings even while it's removed from the catalogue.

- The shape is a **maintainer hierarchy** (think Linux/BDFL): a steward over an open

base, intended to widen its operator and moderator set as the network grows.

---

14. Scaling — what the bandwidth actually costs

Because every audio byte is **relayed** (`seeder → mini-node → listener`), each active stream costs a mini-node **2× the flow rate** — once on ingress, once on egress.

```
one active stream at the 4 Mbit/s cap  =  4 in + 4 out  =  8 Mbit/s on the relay
```

- **Measured baseline:** one capped stream ≈ 4 Mbit/s in + 4 Mbit/s out ≈ 8 Mbit/s.
- **Capacity (≈1 Gbit/s relay NIC): ~125 concurrent capped streams** before

saturation. Steady *playback* (song bitrate ~256 kbit/s, not the 4 Mbit/s burst) is ~0.5 Mbit/s/stream — so **~2,000 steady streamers** fit on one relay.

- **For 200 users:** one ~1 Gbit mini-node comfortably serves steady streaming

(~100 Mbit/s peak); the only stress is many *concurrent downloads* bursting at the cap. The fix is horizontal: players load-balance across a **mesh of mini-nodes** — budget roughly **one 1 Gbit relay per ~125 concurrent capped flows**, far more for steady streaming.

- The real recurring cost is **metered data**, not peak Mbit/s: roughly *2× song

size per play* (in + out). At ~8 MB/song, 200 active users at 20 plays/day ≈ **~1.9 TB/month**; at 50 plays/day ≈ **~4.8 TB/month**.

CPU and memory are not the constraint at these scales — bandwidth is.

---

15. Decentralization posture

Bopwire is **decentralized where it can be and centralized only where a public process would be unsafe** — and it keeps those layers separate on purpose:

- **Decentralized:** content distribution (the swarm), discovery (private DHT +

SwarmIndex), value transfer (self-signed transactions), and token issuance (fixed, algorithmic, minted by genuine plays — no pre-mine, no sale, no protocol fee to any central party).

- **Centralized by necessity:** identity verification and escrow *release*, and

content takedown. These can't be a public vote without becoming an attack surface (fake-identity approval, or a mob withholding an artist's money). This is an **authorization** role — the chain holds the funds; moderators only *attest* and *can't divert* — not a custodial one.

- **Stewardship:** development and root authority currently rest with a founder

operating as a node on the *same terms as any operator* (no special token reward, no fee, no allocation). The intended trajectory is the Linux model — a steward atop a widening, independent base of node operators and moderators, with issuance immutable underneath.

The design goal is a network whose **value is created by its participants and recorded on a ledger no one privately owns**, with the unavoidable trust (identity, takedown) confined to a narrow, accountable, non-custodial role.

This document describes the system as implemented across the `bopwire` (C++ node/chain), `librats` (P2P transport), and `bopwire_player` (cross-platform client) components. It is a technical specification, not an offer, solicitation, or financial instrument.