Horriya v1.0 Roadmap (Go edition, toward “100/100”)

How Horriya works (plain English)

- **Everyone runs the app = a node.** Your node has a keypair (your ID) and an append-only log of your actions (posts, votes, follows). Each action is signed by you.
- **Nodes “announce” new items with tiny messages** (hash + metadata) over gossip. Peers who care fetch the real content by its hash.
- **Content is stored by many nodes.** Authors and followers pin a chunk of posts; other peers may also store pieces according to a lightweight replication rule. If some peers go offline, others still have the bytes.
- **Ranking is local.** Each node computes post rankings using votes and a time-decaying “influence” (Reddit-ish). No blockchain, no global order.

Answers to your questions

1) **When a node joins** - It generates (or loads) an **ed25519** keypair. - It dials a few **bootstrap peers** (multiaddrs you ship), does **mDNS** on local LAN, and joins **GossipSub** topics. - It advertises its interests and follows, then starts receiving small **announcements** (CIDs) for new posts/votes. It fetches only what it needs.

2) **How content is shared** - Only tiny **announcements** are broadcast. - If you care about an item, you **request the block by CID** from one or more providers (peers who hinted they have it). - Your node **verifies** signatures and hash, stores the block in a **BadgerDB** content-addressed store, and updates local indices.

3) **Are a node’s posts only stored on it?** - No. The **author pins** recent posts; **followers pin** some window of posts; plus **rendezvous-hash replication** selects extra holders from currently connected peers. Result: multiple independent copies without central servers.

4) **Is there a periodic health/repair algorithm?** - Yes. A lightweight **replication auditor** runs periodically: - Sample tracked posts; check **provider count** (from gossip hints; when DHT is enabled, also via DHT provider lookups). - If providers < target **R**, the node **re-announces** the CID and (if it has space and is selected by rendezvous hashing) **pins** or **reseeds** missing chunks (with erasure coding in later milestones).

5) **How do peers discover each other?** - **Early:** hardcoded **bootstrap multiaddrs** + **mDNS** on LAN + optional **manual add** (paste a multiaddr) + a few volunteer **libp2p relays**. - **Later:** add **Kademlia DHT** (S/Kademlia hardened) for wide-area peer and provider discovery.

6) **If Kademlia comes later, how do nodes start and how is data stored first?** - **Start:** nodes connect via the bootstrap list, relays, and mDNS; that’s enough to form a gossip mesh. - **Store:** authors/followers pin; extra replicas are assigned via **rendezvous hashing among connected peers**, and **provider hints** are gossiped so fetchers know whom to ask—no DHT required to begin.

Phased roadmap (Go)

Each milestone has Goal, Deliverables, and DoD (definition of done). We keep the "announce → fetch by CID" design from day one to avoid throw-away work.

Phase A — Foundations (local, deterministic)

Milestone A1 — Keys, Signed Author Log, Local Store - **Goal:** ed25519 identities; per-author append-only log; verify signatures and sequence; persist locally. - **Deliverables:** - Module `core` with packages: `crypto` (ed25519), `types` (AuthorEvent, Post, Vote, Follow), `store` (Badger-backed blockstore), `cid` (go-multihash/go-cid helpers). - Protobuf schemas (`/proto`) + codegen with `protoc` or `buf`. - CLI: `horriya keys new`, `horriya post "text"`, `horriya feed --author <pk>`. - **DoD:** Unit tests for sign/verify, seq chaining, CID stability.

Milestone A2 — Local Ranking Pipeline - **Goal:** Influence math (time decay, `log2(1+I)`), vote budgets, Reddit-style hot/Wilson; deterministic replay from log. - **Deliverables:** `rank` and `influence` packages; offline replay tool. - **DoD:** Same input log → same top-N across runs.

Phase B — Networking basics (gossip announce, fetch on demand)

Milestone B1 — Transport + LAN - **Goal:** libp2p (QUIC + Noise), ping/identify, **mDNS** discovery, manual peer add. - **Deliverables:** package `net`; CLI `horriya peers`. - **DoD:** two local nodes exchange pings and identities.

Milestone B2 — Gossip announcements - **Goal: GossipSub** topics (`author/<pk_prefix>`, `discovery/<shard>`), dedup by CID, size-capped envelopes. - **Deliverables:** `announce` type {cid, author, seq, ts, kind, size_hint}. - **DoD:** follower receives announcements p95<2s on LAN.

Milestone B3 — On-demand block fetch - **Goal:** `Get(CID)` over libp2p stream; provider LRU; **hedged** parallel fetches. - **Deliverables:** package `rpc` with `Have/Get/ListProviders`. - **DoD:** after announce, follower fetches & verifies block.

Phase C — Social graph, votes, discovery

Milestone C1 — Follow/Unfollow - **Goal:** Follow events; shard subscriptions; following feed. - **DoD:** UI/TUI shows followed authors in time order.

Milestone C2 — Votes + online influence - **Goal:** Vote events, token-bucket budgets, incremental influence updates. - **DoD:** Influence invariants tested; online rank = offline replay.

Milestone C3 — Discovery + Admission PoW + VRF gate - **Goal:** Public feed without spam; small **Hashcash** for discovery posts; **VRF** lottery to throttle visibility probability by influence. - **DoD:** Floods from zero-influence accounts don’t surface without PoW and endorsements.

Phase D — Scale & availability

Milestone D1 — DHT (S/Kademlia) + bootstrap - **Goal:** WAN provider lookups and robust peer discovery. - **DoD:** Nodes behind NAT can discover/fetch posts via DHT/relays.

Milestone D2 — Rendezvous replication + quotas - **Goal:** Even replication; storage quotas & eviction; target provider count **R** per post. - **DoD:** Measured redundancy ≥R for hot posts.

Milestone D3 — Erasure coding + repair - **Goal:** n-of-k chunking for media; background repair when provider count drops. - **DoD:** Posts recover despite partial loss.

Phase E — Privacy & abuse hardening

Milestone E1 — Dandelion++ (stem/fluff announces) **Milestone E2 — Peer scoring + EigenTrust-lite** (feeds into gossip mesh, provider choice, replication slots) **Milestone E3 — Tor/I2P pluggable transports**

Phase F — UX & multi-platform

Milestone F1 — Desktop app (Go + Wails or WebUI) Milestone F2 — Mobile (Gomobile bindings or Kotlin/Swift SDK over a Go core via cgo)

Phase G — Simulation, SLOs, upgrades

Milestone G1 — 5k–50k node simulator (Go + containerized netem) Milestone G2 — Protocol negotiation & compatibility (multistream-select, multicodec)

Phase H — Release candidate

Security review, fuzzing, packaging, onboarding wizard.

Go tech choices & starter skeleton

- **Language/runtime:** Go 1.22+
- **Crypto:** `crypto/ed25519`
- **Protobuf:** `google.golang.org/protobuf`, `buf.build` optional
- **Content addressing:** `github.com/multiformats/go-cid`, `github.com/multiformats/go-multihash`
- **Storage:** `github.com/dgraph-io/badger/v4`
- **P2P:** `github.com/libp2p/go-libp2p`, `.../pubsub` (GossipSub), `.../p2p/discovery/mdns`, `.../p2p/discovery/routing`
- **DHT:** `github.com/libp2p/go-libp2p-kad-dht`

Repo layout

```
horriya/
  go.mod
  proto/author.proto
  cmd/horriya/main.go
  internal/
    core/    (types, cid helpers, signing, blockstore)
    rank/    (influence, scoring, budgets)
    net/     (libp2p host, gossip, rpc, mdns, relays)
    dht/     (added in D1)
    repair/  (replication auditor)
    ui/      (basic TUI/HTTP UI later)
```

Minimal code sketch (A1)

- `internal/core/keys.go` — generate/load ed25519 keys (PEM or raw).
- `internal/core/types.proto` — AuthorEvent, PostBody, Vote, Follow.
- `internal/core/blockstore.go` — Badger key = CID bytes; value = protobuf bytes.
- `internal/core/feed.go` — append verifies `seq == prev+1`, sets `prev_cid`, computes CID.
- `cmd/horriya/main.go` — subcommands: `keys new/show`, `post`, `feed`.

Libraries you'll read as you go

- go-libp2p docs & GossipSub spec
- go-libp2p-kad-dht examples
- multiformats (CID/multihash)
- BadgerDB docs
- For VRF: ristretto/curve25519 VRF (e.g., Cloudflare's or Key Transparency VRF) — add at C3.

Next steps (let's code Milestone A1 in Go)

1) Initialize module and deps:

```
go mod init github.com/you/horriya
go get github.com/dgraph-io/badger/v4
       github.com/multiformats/go-cid
       github.com/multiformats/go-multihash
       google.golang.org/protobuf
       github.com/spf13/cobra
```

2) Define `proto/author.proto` (same fields as before) and generate Go types. 3) Implement `internal/core` (keys, CID helpers, blockstore, feed). 4) Wire a minimal CLI with Cobra to create keys, write a post, and print your local feed.

When you're ready, I'll draft the Go files for **A1** (keys, types, blockstore, feed, CLI scaffold) so you can run `go build` and post your first signed event locally.