Depined v2

DPN-WP-002 TECHNICAL WHITEPAPER

REVISION 2026.11

Depined v2

A compute, bandwidth and storage network run by its members

Depined turns ordinary computers into one network that can serve AI models, crawl the web, relay traffic and store files. This paper describes how it is built: the member node, how work is scheduled and checked, how members talk to each other, how files are split and rebuilt, and how every layer is encrypted and verified.

APPLIES TO

Desktop node 0.5.x · Control plane 2026.11

PLATFORMS

Windows · macOS · Linux · Browser extension

AUDIENCE

Members, builders, security reviewers

01 Design goals

Depined has to work on the computers people already own: laptops behind home routers, gaming PCs with consumer GPUs, Macs, office machines that sleep at night. Everything below follows from five requirements.

- **Any hardware.** A workload describes what it needs. Each member runs the version of it that fits its own hardware, whether that is an NVIDIA, AMD, Intel or Apple GPU, or just a CPU.
- **No open ports.** Members only make outgoing connections. Nobody has to change router settings, and nothing on a member's machine is reachable from the internet.
- **Private by default.** Members never learn each other's addresses unless they work together on the same model. Stored files are encrypted before they leave the network edge.
- **Verify, then pay.** Points for work are held as pending until the work has been checked. A member that returns wrong answers loses its pending points and trust.

- **Nothing runs unless it is signed or pinned.** App updates, container images and model files are all checked against a signature or a content hash before use.

02 Architecture

The network has three layers. **Members** provide the resources. The **Depined Hub** is a globally distributed edge service that holds one encrypted session with every member and routes traffic to them. The **control plane** decides who does what: accounts and points, the planner, the workload catalog, validation, storage placement and the network directory. Three supporting services complete it: the **Depined Relay** carries encrypted peer-to-peer traffic when two members can't reach each other directly, the **Depined Registry** serves container images, and **Depined Releases** serves signed app builds.

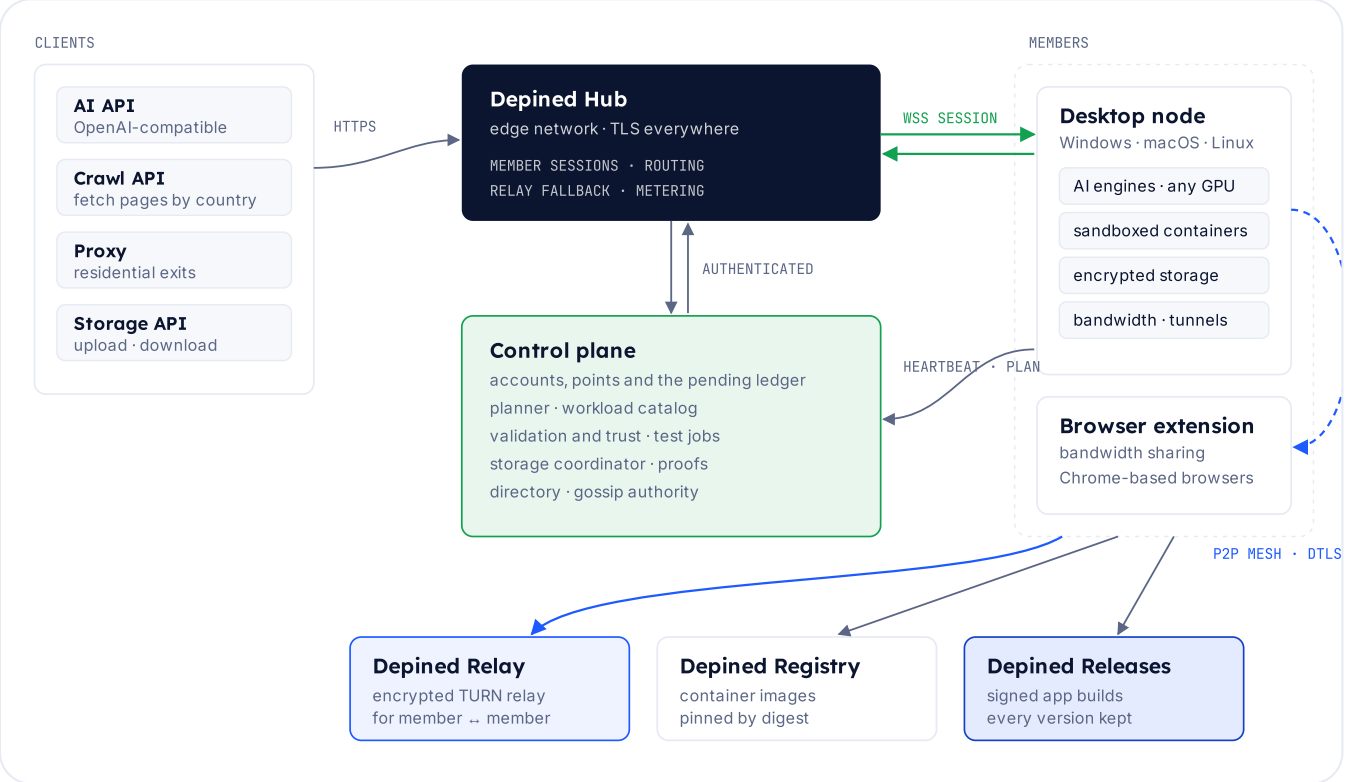


FIG. 1 System overview. Members only open outgoing connections: one session to the hub, short requests to the control plane, and encrypted links to selected peers. Nothing on a member's machine listens on the internet.

COMPONENT	ROLE	TALKS TO
Desktop node	Runs AI engines, containers, storage and bandwidth sharing. Single Go binary, no installer dependencies.	Hub (session), control plane (plan, jobs), peers (mesh)
Browser extension	Shares bandwidth for the crawl API and proxy.	Hub
Depined Hub	Holds member sessions, routes crawl, proxy and API traffic, meters usage once a minute, relays member traffic as a last resort.	Members, control plane
Control plane	Accounts, points, planner, workloads, validation, storage coordinator, directory, gossip certificates.	Hub, members
Depined Relay	TURN relay for encrypted member-to-member links. Credentials are short-lived and issued per member.	Members
Registry and Releases	Static, read-only stores for container images and app builds.	Members (download only)

03 The member node

The desktop node is one self-contained program for Windows, macOS (Apple Silicon and Intel) and Linux (x86-64 and ARM64). It shows its window through the system browser engine, lives in the tray or status bar, starts at login and keeps running when the window is closed. Members choose what to share: bandwidth, CPU, GPU, storage and split models, each with its own switch.

What a node can do

AI ENGINES

llama.cpp · stable-diffusion.cpp

Vulkan, CUDA, Metal or CPU, picked per machine by probing what actually works

CONTAINERS

Docker-compatible

Docker, Podman, OrbStack, Colima; NVIDIA GPUs inside containers

STORAGE

Encrypted pieces

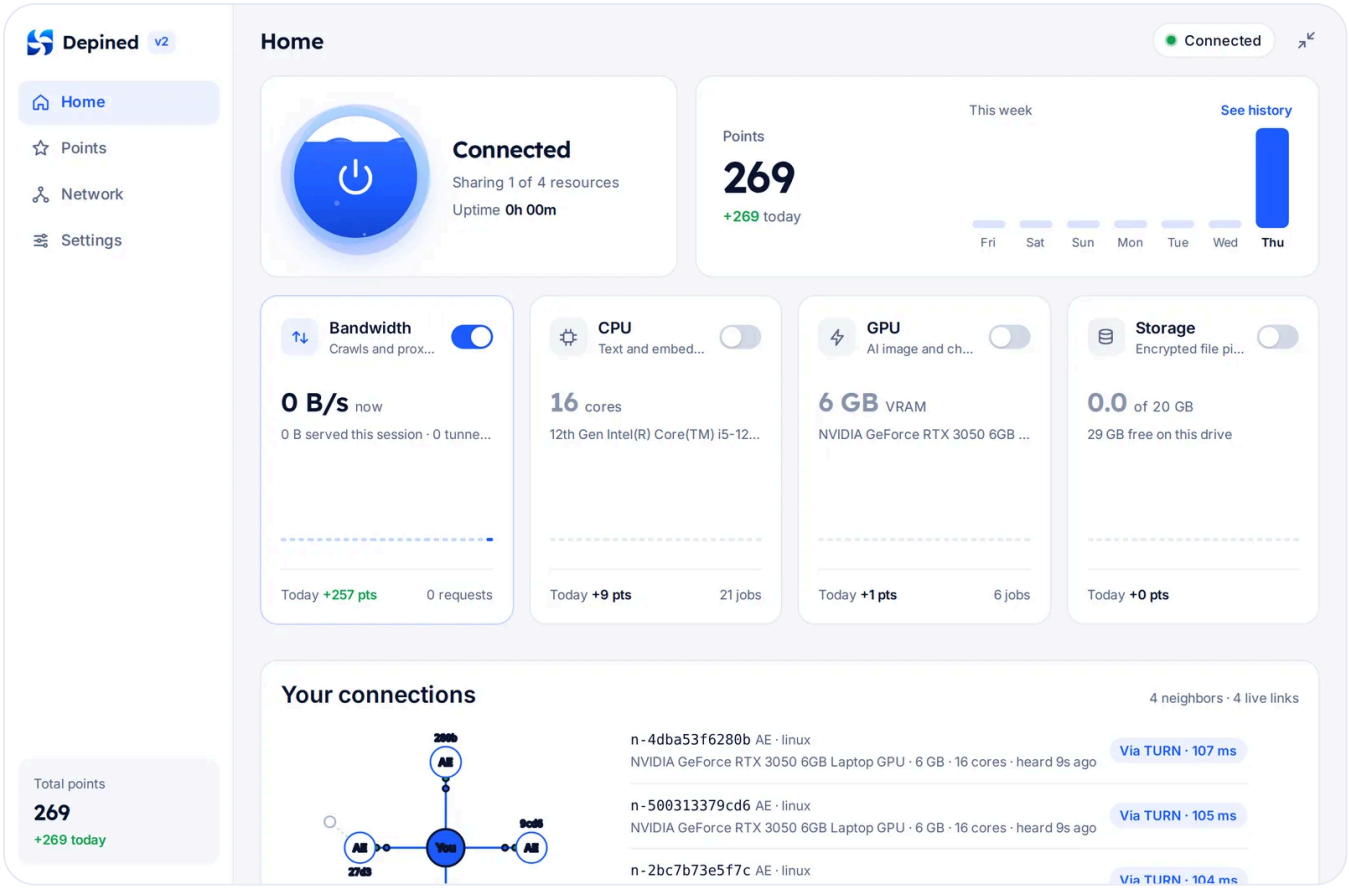
an embedded IPFS node holds erasure-coded ciphertext

BANDWIDTH

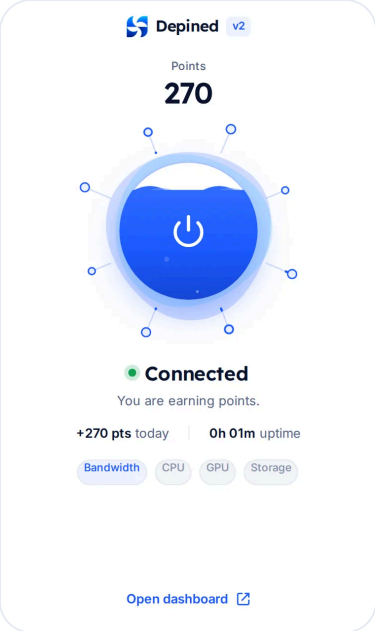
Fetches and tunnels

public web only; the member's own network is blocked

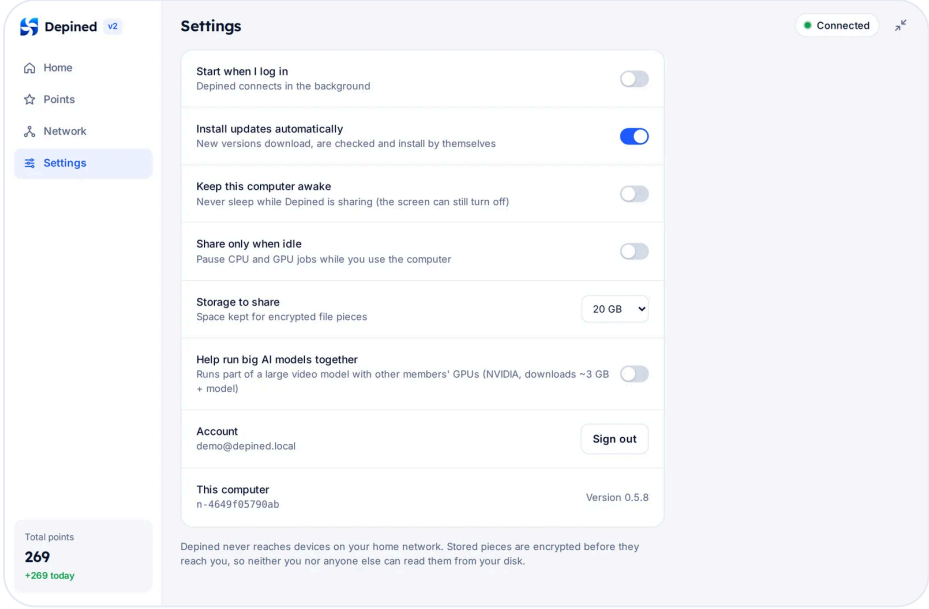
On first start the node checks the machine. On Linux, if containers or GPU support are missing, it installs the container runtime and GPU toolkit itself after a single administrator password prompt. It can also keep the computer awake while sharing, pause work while the member is using the machine, and free disk space for models it no longer needs.



SCREEN 1 The desktop app. One switch to start sharing, points for the week, and a card for each resource with its live rate and what it earned today. Hardware is detected automatically.



SCREEN 2 The compact widget for the tray.



SCREEN 3 Settings: start at login, automatic signed updates, keep awake, idle-only sharing, storage size and split models.

Every 30 seconds the node sends a heartbeat describing its hardware (GPU name and memory, cores, RAM, free disk, which GPU paths work), what it shares, which models are on disk and which are ready to take jobs. The answer is its **plan**: the list of models and workloads it should hold, with everything needed to run them.

---

04 Workloads and scheduling

A **workload** is anything the network can run. It has a kind and an API shape:

- `job` : runs once per request and exits, for example a batch transform.
- `serve` : a long-running model server that answers chat, embeddings, image or video requests.
- `group` : one model split across several members (see the next section).

Each workload lists ordered **variants**. A variant is either a container image (CUDA or CPU) or a native engine with its model files (GPU or CPU). The planner gives every member the first variant its hardware can run, so the same workload is served by a CUDA container on one machine, Vulkan llama.cpp on an AMD laptop, Metal on a Mac and the CPU build on an office PC.

```
{
  "id": "qwen3-4b", "kind": "serve", "api": "chat",
  "variants": [
    { "runtime": "native", "engine": "llama", "accel": "gpu", "vramMB": 4000,
      "files": [{ "url": "…/Qwen3-4B-Q4_K_M.gguf", "sha256": "7485fe6f…", "sizeMB": 2381 }]
},
    { "runtime": "native", "engine": "llama", "accel": "cpu", "files": ["…same file…"] }
  ]
}
```

Clients call the network with the standard OpenAI routes (`/v1/chat/completions` , `/v1/embeddings` , `/v1/images/generations`), using the workload id as the model name. Requests go to members that already have the model loaded, so the first token doesn't wait for a download. A request that takes longer than about 90 seconds returns a job id to poll. Jobs carry the workload version, so a member never receives a job for a version it doesn't hold.

---

05 Split models

Some models don't fit in one consumer GPU. A `group` workload splits the model's transformer blocks across several members. One member is the **lead**: it holds the text encoder and the first blocks, takes the job and drives each step. The others are **partners** that each hold a range of blocks. Activations pass between them over the peer-to-peer mesh.

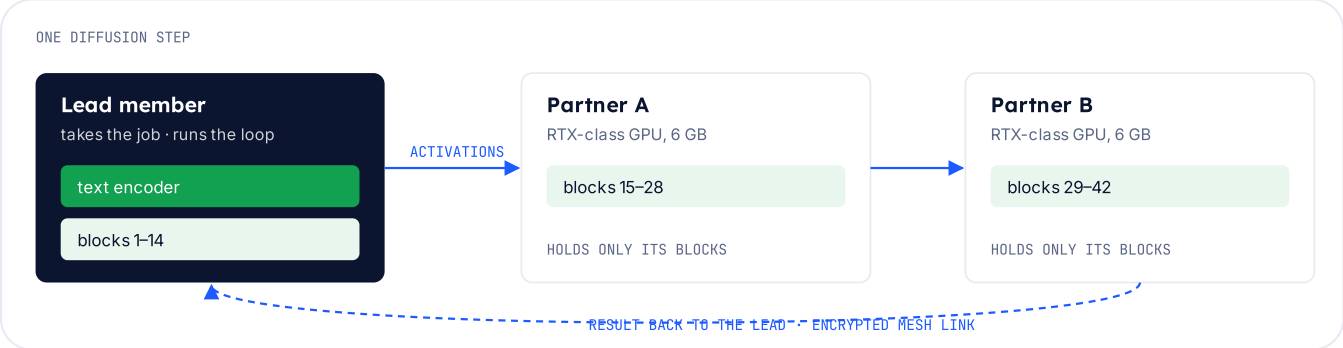


FIG. 2 A model three times larger than any member's GPU memory runs across three members. The planner sizes each partner's block range to the GPU memory it reports.

The planner forms groups from members that share split work, sizes each block range to the partner's GPU budget, and pays every partner a share of the job's points. Partners keep their links to each other warm. Only members of the same group may send each other relay requests or open direct connections, and the control plane only tells a member who its partners are if that member is itself in the group.

06 Peer-to-peer mesh

Members talk to each other over WebRTC data channels. Every channel is encrypted with DTLS and carried over SCTP. Connection setup messages travel through the hub. The data itself takes the best path available:

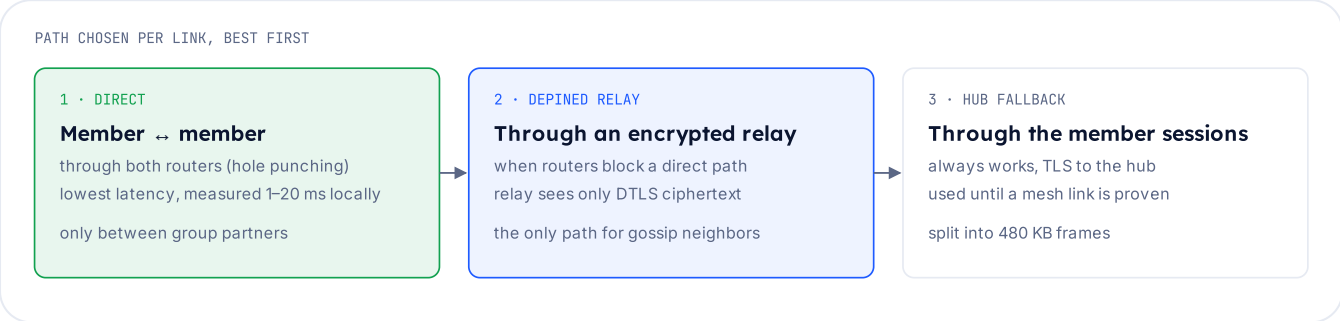


FIG. 3 The connection ladder. A link only carries data after it has answered a ping. If pings go unanswered for 45 seconds the link is dropped, traffic moves back to the hub and the link reconnects on a later round.

Two details make the mesh reliable on real home networks. First, **liveness**: a channel can look open while nothing gets through, which we observed on relayed links. Each link is pinged every 20 seconds, the round trip is measured and shown in the app, and a link is trusted only after it has answered. Second, **address hygiene**: virtual interfaces (Docker bridges, VPN and VM adapters) and link-local addresses are never offered as candidates, because they cause connection attempts to stall on machines that run containers.

Large transfers use flow control: the sender waits while the channel's buffer is above a high-water mark, so a multi-gigabyte model shard never sits in memory. Messages are split into 60 KB chunks and reassembled on the other side, and duplicates are dropped when a message is resent over the hub.

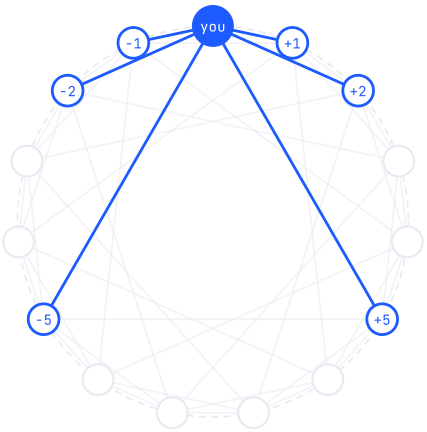
07 Gossip network

Every member should know who is online and what they have, without having to ask a server. Members therefore keep links to a few **neighbors** and pass signed status records along those links. News reaches the whole network in a few rounds, and the app shows each member its neighbors on a live map.

Choosing neighbors

The control plane places members on a ring, ordered by a hash of their id. A member's neighbors are the members one and two places to either side, plus one a third of the way around the ring. The

offsets are symmetric, so if A is B's neighbor then B is A's neighbor. Each member has at most six neighbors regardless of network size, and the long chords keep the distance between any two members short.



RING ORDER
members sorted by a hash of their id

NEIGHBORS
±1, ±2 and ±⌊N/3⌋ places: at most 6, always symmetric

RECORDS
signed every 15 s, sent every 10 s, kept 3 min

LINKS
relay only, encrypted, addresses never exchanged

FIG. 4 A 15-member ring. The highlighted member links to six neighbors: ±1, ±2 and ±5 places. Every member does the same, so a record reaches any other member in at most three hops.

Signed records

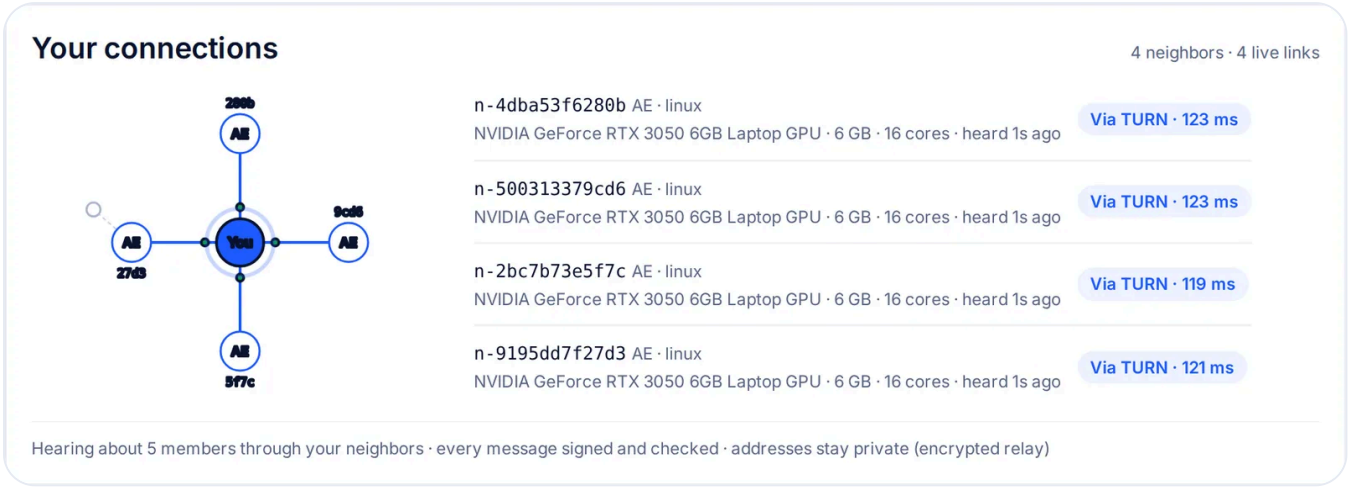
Each member generates its own Ed25519 key on first start. The control plane signs a **certificate** binding the member id to that key, valid for two hours and renewed every minute. Every 15 seconds a member signs a fresh record about itself, containing its hardware, which models are ready, whether it is busy, its app version and its links. Every 10 seconds it sends each neighbor the records that neighbor hasn't had from it yet.

A receiving member accepts a record only if all five checks pass:

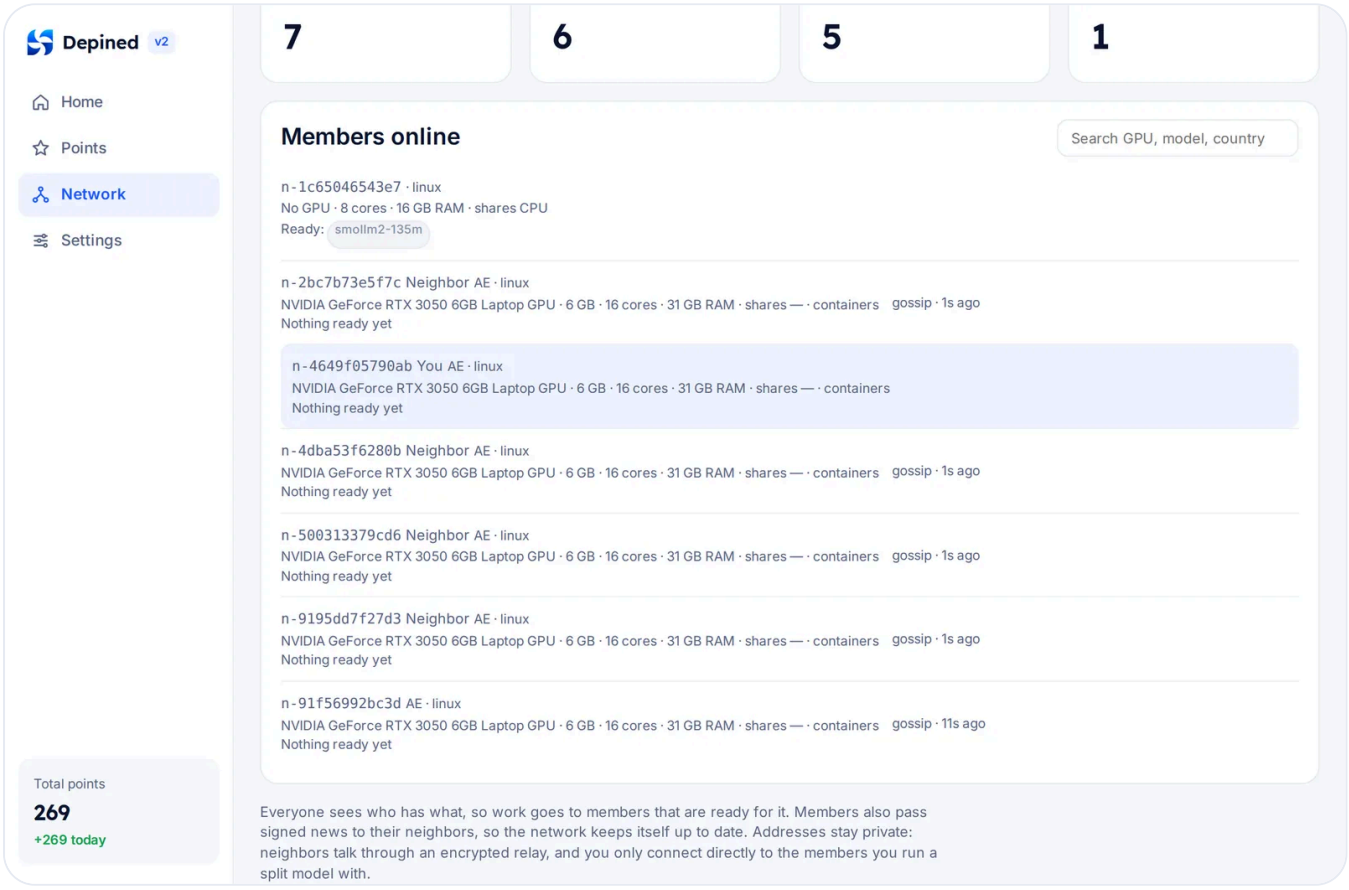
CHECK	REJECTS
Certificate signed by the network authority	members invented by an attacker
Certificate names the record's member	a member speaking for another member
Certificate not expired	keys of members that were removed
Record signed with the certified key	records changed in transit, for example a GPU claim edited
Record at most 3 minutes old	old news replayed to make a gone member look alive

Gossip is only accepted from current neighbors or group partners, at most one message every two seconds per sender, capped at 1 MB and 400 records. Links to neighbors are forced through the Depined Relay. Both sides gather only relay candidates and drop any other candidate they receive, so a neighbor never learns a member's IP address. In a six-member test network, every member knew all

five others within 16 seconds, with zero rejected records and 24 of 24 neighbor links running through the relay.



SCREEN 4 "Your connections" in the app. You are in the middle, your neighbors sit around you and the member they told you about sits further out. Dots moving along a line are signed records being exchanged, and each row shows the link type and round-trip time.



SCREEN 5 The Network page: who has what across the network, with neighbors tagged and how long ago each member was last heard through gossip. No IP addresses are shown.

WHY GOSSIP ON TOP OF A DIRECTORY

The control plane still publishes a directory of who has what. Gossip makes that knowledge independent of it: members exchange it directly, keep it fresh every few seconds, and the app falls back to the gossip view if the control plane can't be reached.

08 Distributed storage

Files stored on Depined are encrypted, cut into redundant pieces and spread across members' disks. No member holds a whole file, and what any member holds is ciphertext.

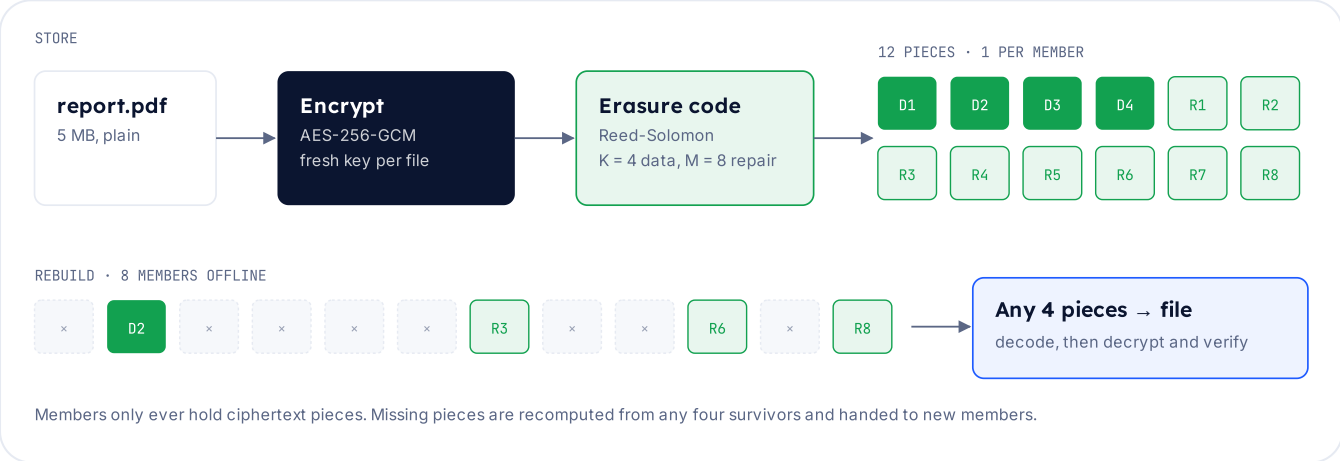


FIG. 5 A 5 MB file: encrypted with its own key, split into 4 data pieces and 8 repair pieces, each piece stored on a different member. Any 4 of the 12 rebuild the file, so it survives losing two thirds of them.

Encrypt, then split

Every file gets a fresh random 256-bit key and is sealed with AES-256-GCM, which both hides the contents and detects any change to them. The file key is itself encrypted with a control-plane secret and never reaches members. The ciphertext is then split with Reed-Solomon erasure coding into *K* data pieces and *M* repair pieces. Any *K* of the *K + M* pieces rebuild it.

FILE SIZE	DATA PIECES K	REPAIR PIECES M	STORED OVERHEAD	PIECES THAT CAN BE LOST
under 1 MB	2	4	3×	4 of 6
1–10 MB	4	8	3×	8 of 12
over 10 MB	10	20	3×	20 of 30

Large files are uploaded in parts. Each part is stored as its own segment with its own key and pieces, so a 10 GB file never has to fit in memory, and downloads stream segment by segment. Each segment is rebuilt from the first *K* pieces that arrive and decrypted as it goes.

Proving storage

Members are paid only for pieces they prove they hold. During a sync a member may be asked for one random block of a piece, answered as `sha256(secret nonce || block)`. The member can only compute it with the block's actual bytes. The expected answers are prepared in advance from the network's own copy and each one is used once, so checks keep working after the network lets go of that copy.

Rebuilding when members leave

The network's own storage node acts as a drop-off point. It keeps a copy only until a file is safe on members: even if the member holding the most pieces disappears, members still online must hold at least *K + 2* different pieces. When members leave and a file falls below that margin, the network fetches pieces back from the remaining members, which is itself a real retrieval check. It then rebuilds the missing pieces from any *K*, refills its prepared checks and hands the pieces out to new members. Pieces are content-addressed, so the same bytes always get the same id and rebuilt pieces slot back in exactly.

---

09 Security and encryption

Depined runs other people's work on members' computers and moves members' traffic across the internet. Both directions need protection: members from the work, clients from dishonest members, and everyone from outsiders. This section lists every protection in place and the threat each one answers.

ENCRYPTION IN TRANSIT

LINK	PROTECTION
Client ↔ hub	TLS. API keys go in headers, never in URLs.
Member ↔ hub	TLS WebSocket. The member authenticates with a signed node pass issued at sign-in.
Member ↔ control plane	TLS with the member's bearer token. The control plane only accepts requests arriving through the hub.
Member ↔ member	DTLS on every WebRTC data channel, both direct and relayed. The relay forwards ciphertext it cannot read.
Relay credentials	Short-lived TURN credentials issued per member by the control plane. No shared static password.
Gossip records	Ed25519 signatures on every record, plus DTLS on the link.

ENCRYPTION AT REST

DATA	PROTECTION
Stored files	AES-256-GCM with a fresh key per file and per segment. Members hold only erasure-coded ciphertext.
File keys	Sealed with a control-plane secret. Never sent to members.
API keys and admin secrets	Stored encrypted (AES-GCM), shown again only to an authenticated admin.
Member signing keys	Generated on the member's machine. The private key never leaves it.

CODE INTEGRITY: NOTHING RUNS UNCHECKED

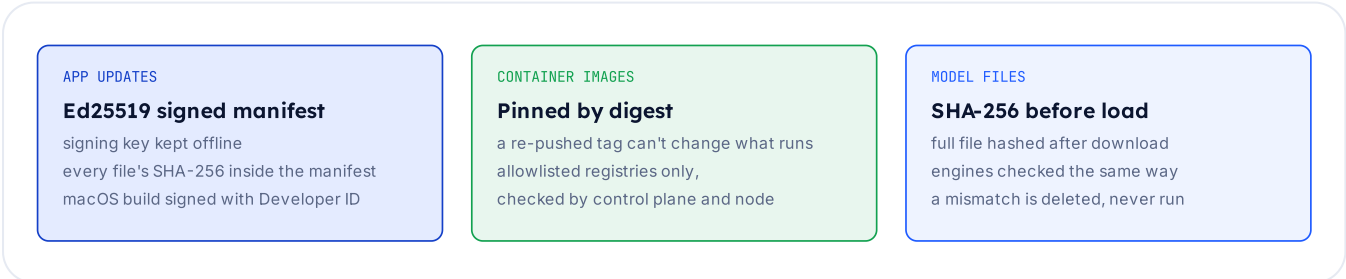


FIG. 6 Three independent integrity checks. A compromised download mirror or registry account can't push code to members, because the expected hashes and signatures come from a different place than the files.

PROTECTING MEMBERS FROM THE WORK

- **Container sandbox.** Every container runs with all Linux capabilities dropped, `no-new-privileges`, and process, CPU and memory limits. Batch jobs run with no network at all unless the workload

needs it.

- **No access to the member's files.** Inputs are copied into the container and outputs copied out. Nothing on the host is mounted.
- **Own containers only.** Every container the node creates is labelled with the node's id, and cleanup only ever touches those.
- **Member's network is off limits.** Bandwidth requests and tunnels may only reach public addresses. The address is checked after DNS resolution, so a public name pointing at a router, NAS or cloud metadata address is refused. Tunnels are limited to web ports (80, 443, 8080, 8443).
- **Gateway token.** Group containers reach the mesh through a local gateway that requires a per-run secret, so other software on the machine can't use it.

PRIVACY

- The network directory lists hardware and ready models, never IP addresses.
- Gossip neighbors connect only through the relay, so they never see each other's address.
- Direct connections, which do reveal addresses, are allowed only between members working on the same split model. The control plane refuses to reveal a group to anyone outside it.
- The app only listens on the loopback interface and requires a local-only header, so web pages can't drive it.

THREAT	WHAT STOPS IT
Member reads stored files	Per-file AES-256-GCM, erasure-coded pieces, keys never sent to members
Member fakes results for points	Pending points, test jobs, cross-checked answers, trust score (section 10)
Member claims storage it dropped	One-time hashed block challenges, points only for proven pieces
Malicious update or image	Signed manifest, digest-pinned images, image allowlist on both sides
Workload attacks the host	Capability-free sandbox, no host mounts, resource limits, no network for jobs
Client reaches the member's home network	Post-DNS private address blocking, port allowlist
Outsider joins a split group	Relay and mesh accept only current partners; group lookup returns 403
Forged or replayed network status	Authority-signed certificates, Ed25519 records, 3-minute freshness
Address harvesting	No IPs in the directory, relay-only neighbor links

KNOWN LIMITS, STATED PLAINLY

When no mesh link is available, member-to-member traffic falls back to the hub. It is TLS-encrypted on each leg but not end-to-end, so the hub could read it. Model-serving containers currently have network access so they can download weights; an egress-filtered network for them is on the roadmap. The control plane holds the keys to stored files, so storage is private from members but not from the operator; client-side encryption is planned.

10 Validated rewards

Members earn points for connected time, bandwidth served, storage proven and jobs completed. Job points are not paid immediately. They go into a **pending ledger** and are confirmed only after a waiting period during which the member's work can be checked.

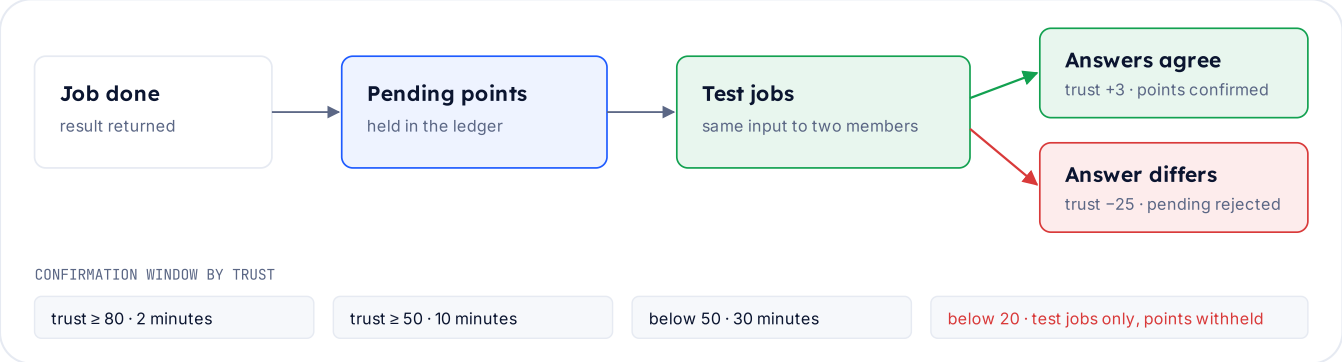
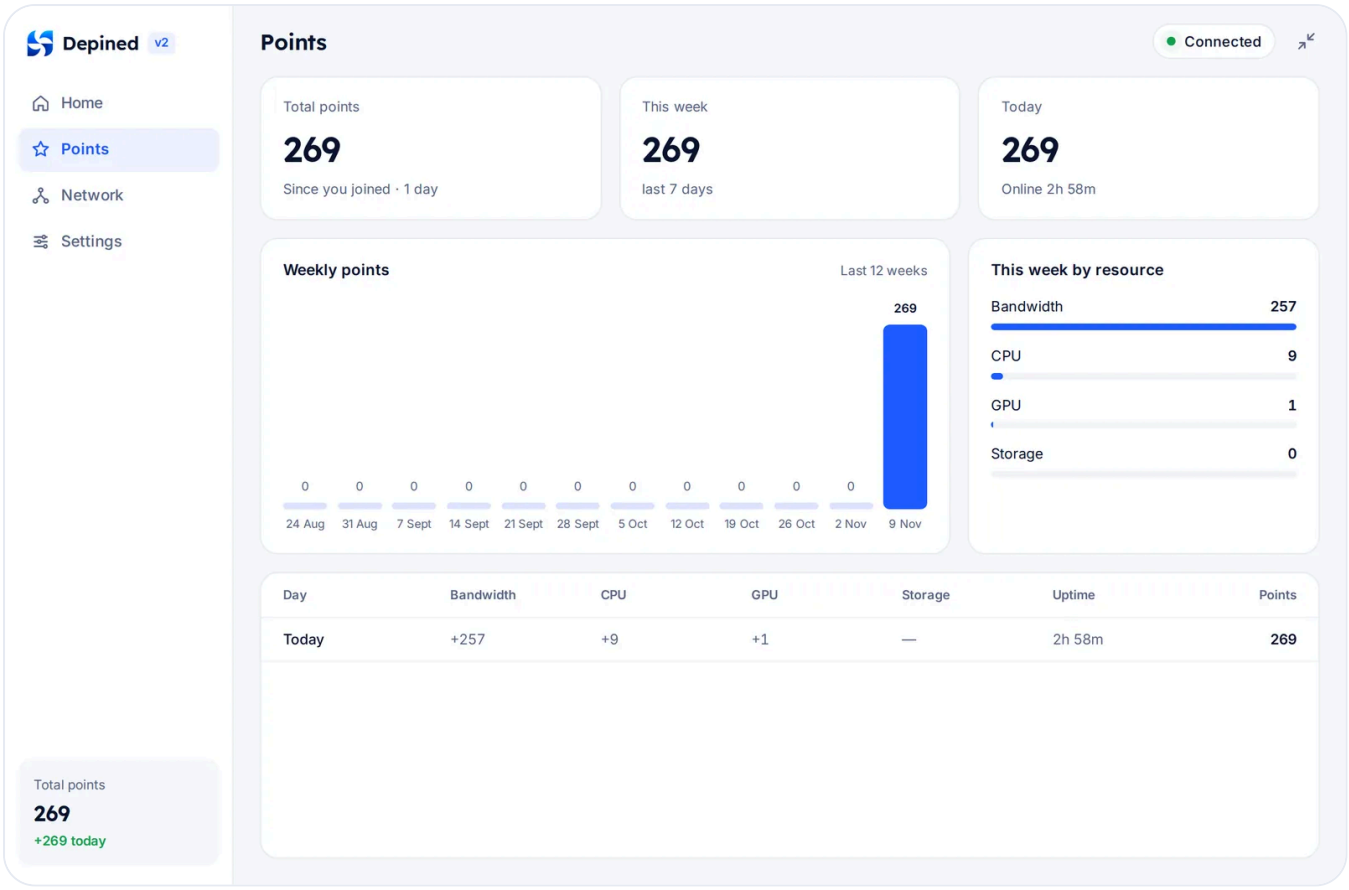


FIG. 7 The reward lifecycle. Every member starts at trust 50. Honest work raises trust and shortens the wait; one wrong answer costs far more than one right answer earns.

Test jobs use fixed seeds and zero temperature, so honest members produce comparable answers. The same input goes to two members, a third breaks a tie, and the agreed answer becomes the reference for later checks. Answers are compared in the way that suits each kind of output:

OUTPUT	COMPARISON	PASSES WHEN
Chat text	Word-level longest common subsequence	`similarity ≥ 0.6`
Embeddings	Cosine similarity	`≥ 0.97`
Images	Perceptual difference hash	`≤ 12 bits differ`
Other jobs	Exact match	`identical`



SCREEN 6 The Points page: totals, the last 12 weeks, this week split by resource, and a day-by-day table.

Test jobs look like any other job to the member, so there is no way to behave well only when watched. In a test with one deliberately dishonest member, the honest members rose from trust 50 to 62 while the dishonest one fell to 25 and lost all its pending points.

---

11 Releases and updates

Releases are built automatically from a tagged commit. The build pipeline compiles every platform, signs the macOS app, writes a manifest listing each file's SHA-256 and signs that manifest with the release key. It uploads the build to Depined Releases under its version number and checks every file through the public address. Only then does it switch the `latest` pointer, putting the manifest up last so no member can ever see a manifest whose files aren't there yet.

Nodes check for updates every hour. A node downloads the new build, verifies the manifest signature and the file hash, replaces itself and restarts. Members can switch automatic installs off and install from Settings instead. Every past version stays available, so rolling back is a matter of pointing `latest` at an earlier release.

```
tag desktop-v0.5.8 → build 5 platforms → sign (Developer ID + Ed25519 manifest)
                  → upload releases/0.5.8/ → verify every SHA-256 publicly
                  → copy to latest/ (binaries first, manifest last) → members update within
the hour
```

12 Roadmap

- **Client-side encryption for storage,** so the operator can't read stored files either.
- **End-to-end encryption for hub-fallback traffic** between members, using the members' own keys.
- **Egress-filtered networking** for model-serving containers.
- **One-click container runtime** bundled for Windows and macOS, matching the Linux installer.
- **Gossip-driven routing,** so members pick partners and job targets from what they hear directly.
- **Signed container images** in addition to digest pinning.

Depined v2 · Technical whitepaper DPN-WP-002 · Revision 2026.11

Figures and thresholds describe the current release and may change.