Zero-Knowledge Proofs for Verifiable Network Performance in DePIN

Currently, Decentralized Physical Infrastructure Networks (DePINs) face a challenge in verifying information. For example, if a given node has been contracted to deliver a certain amount of data at a certain speed/latency, how can the network prove that it was delivered without trusting the node's own reports or a centralized speed test server. Whether this type of problem occurs in digital resource based DePIN (like computing, storage, or AI Inference) or physical resource based DePIN (like sensors), these networks either leak sensitive logs, rely on trusted nodes, or assume specialized hardware. In 2019, Vitalik noted that there was no clean, after-the-fact cryptographic proof of bandwidth, unless you add interaction and extra assumptions. This issue still continues in recent times, showing up in the real-world. In early 2024, fraudulent actors sent fake uptime data to IO.net to earn rewards, falsifying the existence of 1.8 million GPUs (2). As of August 2024, an op-ed on Coindesk showed that unverified GPUs on IO.net outnumbered real GPUs 3-to-1.(3)

Similarly, Helium's early development saw cases of falsified node locations, clustered deployments, and mutual attestations, which all hurt the authenticity of the network.(4)

Current Challenges with DePIN

Existing DePIN verification methods are often fragile, with many verification methods lacking sufficient trust, privacy, and accuracy, as evidenced by the aforementioned GPU spoofs and network falsification. Without authentication and robust checks, nodes or service providers may receive payouts on false pretenses. This can result in networks that are more vulnerable to gaming, causing users to adopt poorer performing/unstable platforms.

Centralized data oracles solve scale by obtaining raw telemetry data and posting a result, but these rely on a single centralized pipeline, leaking operational data (routes, timestamps, counterparties) to that oracle. This often defeats the purpose of a decentralized infrastructure system completely. Helium's shift to off-chain oracles is a clear example of this, as it increased throughput, but required validation from privileged services. (9) (As opposed to portable cryptographic proofs any third party can verify independently).

Another currently used solution to this problem is multi-party "speed tests." Research like proof of backhaul or proof of bandwidth coordinate challengers to probe a node and estimate capacity. This reduces reliance on a single tester and makes collusion harder, but still results in logs which reintroduce trust and leak sensitive traffic patterns. (5, 6)

Why Not Use TEEs

TEEs, or Trusted Execution Environments, aim to create secure computing platforms on the hardware level. But while they may seem to offer hardware-guaranteed verification, they face major limitations in DePIN contexts. In practice, TEEs are vulnerable to side-channel attacks, require the user to trust a chain of systems (keys, firmware, updates), and still expose raw logs to the operator.(7, 8) TEEs are simply not a sufficient root of trust for DePIN verification.

Implementing a Zero Knowledge Framework

In this case, zero-knowledge proofs offer a mathematically robust alternative to prior systems, allowing nodes to prove they meet service-level objectives without revealing traffic or measurement transcripts. Conceptually, instead of uploading the actual logs to some source, the node would publish a small proof which can verify the following things:

1. All tests it claims to have passed were authentic (matched results from independent testers)
2. The totals/thresholds (bandwidth/latency) were satisfied

Anyone can check this proof later, allowing for verification of accuracy without requiring any information about the system itself.

With zero-knowledge in this type of framework, ZKPs would be generated locally on compute nodes, ensuring proof data remains in the hands of the network participants without centralizing operations. As a quick example of how this could be implemented, a DePIN system could do the following:

1. Set up tests: Independent verifiers publish commitments to a prebuilt set of test packets (hashed). This makes sure that a node can't claim to have passed a nonexistent test.
2. Run challenges: At random times, these verifiers can send "new" (to prevent precomputation) packets to the node.
3. Node processing: The node processes the challenges, and computes the answer/timestamp. (Note: this isn't uploaded)
4. ZKP: The node then generates a zero-knowledge proof that checks authenticity, correct responses, and whether they met totals/thresholds.
5. Verify/Pay: Once a smart contract verifies the proof is correct, it can instantly execute the contract for payment, ensuring the performance/security of the DePIN node without revealing any data.

Comparison to Existing Systems/Networks

One network currently working this type of system is NovaNet, which positions itself as a ZK proving network/zkVM that partners with DePIN projects such as IoTeX and io.net for device identity/performance checks. (10, 11). But while the infrastructure is useful, NovaNet doesn't define the application-level framework that is required; we need an open measurement protocol for end-to-end bandwidth/latency and a standard verifier that anyone can use to check output without logs. There's also the inherent privacy risk that occurs when weak devices outsource proof generation, a well-documented problem and risk (12). By instead using a locally created ZKP system which is verified on chain, it ensures information never leaves nodes, preserving privacy and information security.

Citations

1: https://vitalik.eth.limo/general/2019/11/22/progress.html
2: https://x.com/shadid_io/status/1784683652342833217
3: https://www.coindesk.com/opinion/2024/08/02/on-device-proofs-solve-depin-verification-challenges
4: https://www.chaincatcher.com/en/article/2179461
5: https://www.ndss-symposium.org/wp-content/uploads/2024-764-paper.pdf
6: https://docs.witnesschain.com/archive/proof-of-bandwidth
7: https://eprint.iacr.org/2023/251.pdf
8: https://sgaxe.com/files/SGAxe.pdf
9: https://docs.helium.com/iot/proof-of-coverage/
10: https://www.novanet.xyz/blog/iotex-and-novanet-bringing-zkps-and-decentralized-id-to-millions-of-depin-devices
11: https://www.coinspeaker.com/io-net-novanet-partner-boost-decentralized-gpu-security/
12: https://blog.icme.io/zk-prover-networks-want-your-data-dont-give-it-to-them-2//

Demo/Example of this System

One interesting way to demo this type of system is with latency benchmarking. Let's imagine that a system wants to prove that it has low latency, or meets a certain threshold. Normally, the computer would have to send logs to a central system (think fortnite servers for ex.) or publish raw timestamp traces so that an auditor can compute Round Trip Times (RTTs). This leaks IP addresses, routes, timing metadata, enables path fingerprinting, and forces trust. Multi party speed tests reduce direct trust but still emit transcripts and information.

Using the ideas defined above, I built a system which proves latencies for DePIN systems without revealing raw RTTs, indices, IPs, or timestamps to any external verifier (and latency information is directly receivable on chain). Only the backend sees private data, and the backend can be built of multiple decentralized oracles or examining systems.

Workflow:

1. **User runs performance test**: At this stage, an end client will receive 32 RTT challenge-response tests from external verifiers. These challenges each must be correctly computed and sent back with latency calculated within 1.5s with 3s grace for validity. For this, I used Next.js websockets and CRAM (Challenge-Response Protocol). For security, I used general networking flow tokens/token-buckets to prevent reuse attacks. (Bound to test → proof → mint)
2. **Merkle Tree:** After the test is complete, the verifying oracles build a merkle tree utilizing the Poseidon hash function for ZKPs. The root of this tree is committed on chain (held as contract made with Solidity + Foundry for tests) after the epoch (L2 rollup on Base Sepolia, all proofs within a certain 5 minute period will be in the same epoch, making it cheaper to query previous calculations). This is temporarily stored in Redis with a deletion timer.
3. **Client Fetches and Proves Tree:** After the oracles/backend create the contract/committed root, the client pulls the leaf bundle (logs are still not on chain). From here, the client generates a zkSNARK proof locally (I coded this in Noir lang with a wasm implementation of barretenberg [I used this for its implementation of the PLONK zkSNARK providing system]), which shows that the merkle root on chain + private RTTs verify correctly.
4. **User submits contracts and mints badge/latency:** After the user finishes the proof, it is submitted to the solidity contract created previously. This contract checks the proof, ensures the merkle root matches, checks if the user has previously minted a proof of latency for the current bucket. If not, the contract executes, minting an "verified" ERC-1155 token which holds the user's wallet address, badges, and latency. (Without ever sharing logs externally). This can then be queried by anyone on chain to directly show a system meets thresholds.
    a. Rainbowkit + Wagmi used for secure wallet connection and minting.

Currently, you can try this system out at **https://zkpbench.vercel.app/**, using a rainbowkit compatible wallet with Base Sepolia. You can get free Sepolia Eth at https://cloud.google.com/application/web3/faucet/ethereum/sepolia, and natively bridge to Base Sepolia at https://superbridge.app/base-sepolia. (Or you can just msg me I'll send some over). Demo video: https://youtu.be/-r4Ho56xJZ4