WHITEPAPER · V1.0 · AUGUST 2026

PulseGrid: real-time, bilaterally trustless machine settlement on BOT Chain

Abstract

PulseGrid is a network that pays physical devices for the work they do, in real time and provably, with no trusted intermediary. Each device signs its own telemetry, a sponsor relayer anchors it on-chain gaslessly, and every settlement interval an autonomous agent pays or slashes each operator in a stablecoin. The decision is not the agent's: the settlement contract takes only a device identifier and an interval and re-derives pay-or-slash on-chain from the device-signed facts and an on-chain policy. The result is a two-way, machine-trustless guarantee: an operator cannot be paid for data it did not deliver, and the agent cannot stiff an operator who did. PulseGrid runs today on BOT Chain mainnet, settling in canonical USDT, with the guarantee backed by an adversarial test suite. This paper specifies the protocol, its security model, and its economics.

1. Introduction

Decentralized physical infrastructure (DePIN) networks pay operators for real-world work: bandwidth, sensing, compute, energy. In practice that payment is almost always retrospective and trusted. An off-chain service tallies "uptime," cuts a payout, and the operator has to trust the tally. The payer can under-report, stall, or dispute, and the operator has no on-chain recourse and no way to prove otherwise. Trust in the payer is the weak point of the entire model.

PulseGrid removes the payer as a trusted party. Each device signs every reading; the network anchors those signatures on-chain; and each interval the settlement contract itself decides pay-or-slash from the signed on-chain facts. There is no invoice to reject, no approver to wait on, and no counterparty who can decide not to pay.

2. Design goals and the guarantee

PulseGrid is built around a single property we call **bilateral machine-trustlessness**:

> An operator cannot be paid for data it did not deliver, and the agent cannot stiff an operator who did.

Everything else serves that property. Concretely, the settlement entry point is `settleEpoch(bytes32 deviceId, uint64 epoch)`. It accepts no amount, no verdict, and no proof from the caller. Point it at any device and interval and the contract re-derives the outcome itself. A buggy or malicious agent therefore cannot overpay, wrongly slash an honest operator, pay an un-anchored interval, or double-settle. Section 6 states these as formal properties and cites the tests that enforce them.

Two further goals shape the design. Devices must be **gasless**: an operator should not need native gas to earn, so devices sign but never transact. And settlement must extend from **liveness** (did you deliver enough?) to **data quality** (was the data in range?), so a device cannot satisfy a policy by streaming garbage.

---

3. Architecture

PulseGrid is five on-chain contracts and a small set of off-chain roles. The contracts hold all authority; the off-chain roles only observe, relay, and drive.

OFF-CHAIN (UNTRUSTED FOR INTEGRITY)

- Device: signs each reading (EIP-712); holds its own key; zero gas.
- Sponsor relayer: submits the anchor and pays gas; cannot forge or alter a reading.
- Settlement Agent / keepers: call settleEpoch each interval; carry no verdict.
- Advisory AI: narrates device health; never touches settlement.

ON-CHAIN (BOT CHAIN MAINNET, 677)

- DeviceRegistry: device to operator and signing key.
- TelemetryAnchor: verifies signatures; counts anchors per interval.
- ServiceContract: the versioned, append-only SLA per device.
- PolicyModule: pure pay/slash verdict library.
- SettlementVault: escrow, bond, keeper pool; settles.

```
device --sign--> relayer --type-3 blob tx--> TelemetryAnchor --counts--> SettlementVault
agent / keeper --settleEpoch(deviceId, epoch)--> SettlementVault <--verdict-- PolicyModule
SettlementVault --USDT--> operator (PAY, from escrow) or treasury (SLASH, from bond)
```

4. The settlement protocol

The protocol is one loop, repeated every interval.

4.1 Sign at the edge

Each reading is signed by the device with EIP-712 typed data. The domain is `{name: "PulseGrid", version: "1", chainId, verifyingContract: TelemetryAnchor}` and the message type is:

```
Telemetry(bytes32 deviceId, uint64 nonce, bytes32 payloadHash, int256 value,
uint64 timestamp)
```

The signed struct covers the reading's `value` as a first-class `int256`, so the numeric reading is as unforgeable as the payload hash. The same typed-data definition is used byte-for-byte on-chain, in the Go device SDK, and in the browser, so one signature verifies everywhere.

4.2 Anchor gaslessly

The device posts the signed reading to a sponsor relayer. The relayer packs the raw payload into an EIP-4844 blob and submits a type-3 transaction that calls `anchorWithBlob(...)`, paying the gas itself. The device spends zero. The relayer is **integrity-untrusted**: it cannot forge the device signature or alter the payload hash, because `TelemetryAnchor` re-checks both. It is trusted only for liveness (it can decline to relay, which simply fails to anchor, which is slashable).

On-chain, `TelemetryAnchor` verifies in order: the signature recovers to the device's registered key (`UnknownDevice` / `BadSignature`); the per-device nonce strictly increases, so replays and out-of-order readings are rejected while gaps are allowed (`StaleNonce`); the device-signed timestamp is within `maxSkew` of block time (`StaleTimestamp`); and, on the blob path, `blobhash(0)` is present (`NoBlob`). Signature recovery rejects high-`s` values (EIP-2 malleability) and requires `v` in 28.

4.3 Bucket by interval

An interval is a settlement epoch. `TelemetryAnchor` derives the governing epoch from the device-signed timestamp (`epoch = timestamp / EPOCH_LENGTH`) and increments two counters: `anchorsInEpoch` for every valid anchor, and `inRangeInEpoch` for anchors whose signed value is inside the governing policy's range. The in-range check reads the SLA that governs that epoch and never reverts, so anchoring liveness never depends on SLA state.

4.4 Settle on-chain

Each interval, the Settlement Agent (or any keeper) calls `settleEpoch(deviceId, epoch)`. The vault:

1. Loads the SLA version that governed that epoch (`slaOfAt`), so terms cannot be repriced retroactively.
2. Requires the epoch to be final: `block.timestamp >= (epoch + 1) * EPOCH_LENGTH + maxSkew + settlementDelay`.
3. Marks the epoch settled **before** any transfer (`AlreadySettled` on a repeat), which is both the one-settlement-per-epoch guard and the reentrancy guard.
4. Reads the on-chain counts and asks `PolicyModule` for the verdict.
5. On PAY, moves stablecoin from the funder's escrow to the operator (capped at the escrow balance). On SLASH, moves from the operator's bond to the treasury (capped at the bond).

---

5. Policy

`PolicyModule` is a pure library and the sole authority on the verdict:

```
function evaluate(SLA sla, uint32 count, uint32 inRangeCount) returns (Action,
uint256) {
    uint32 qualifying = sla.checkValue ? inRangeCount : count;
    if (qualifying >= sla.minAnchors) return (PAY, sla.pricePerEpoch);
    return (SLASH, sla.slashPerEpoch);
}
```

An SLA is a versioned record: `funder`, `minAnchors`, `pricePerEpoch`, `slashPerEpoch`, `checkValue`, `minValue`, `maxValue`, and the `activeFromEpoch` it takes effect. `setSLA` is append-only and monotonic, may be called only by the device's operator or the incumbent funder, and forces `activeFromEpoch` to be strictly greater than the current epoch. That single rule is what makes the value layer sound (Section 7): the policy governing an in-progress epoch is frozen before the epoch begins, so the range applied while counting anchors is exactly the range settlement re-derives.

---

6. The bilateral-trustless guarantee

Because `settleEpoch` carries only `(deviceId, epoch)` and the vault re-derives everything, the following hold for any caller, honest or malicious. Each is enforced by a named test in the suite.

- **No overpay.** PAY is exactly `pricePerEpoch`, capped at escrow. `test_Adversarial_AgentCannotOverpay`.
- **No wrongful slash.** A device that met the policy is paid, never slashed. `test_Adversarial_AgentCannotWronglySlashHonestOperator`.
- **No pay for un-anchored intervals.** An interval with no qualifying anchors slashes, never pays. `test_Adversarial_AgentCannotPayUnanchoredEpoch`.
- **No double-settle.** The first settlement wins; a second reverts. `test_NoDoublePay_SecondSettleReverts`, `test_KeeperRace_LoserRevertsAlreadySettled`.
- **No pay on a forged anchor.** A bad signature never anchors, so the epoch stays slashable. `test_Adversarial_ForgedAnchorRejected_EpochStaysSlashable`.

The suite has **81 passing tests** in total, including fuzz tests that pay if and only if the count meets the minimum, and value-layer tests that pay if and only if enough readings are in range.

---

7. Value-layer SLAs (data quality)

Liveness alone lets a device satisfy a policy by streaming anything. A value-layer SLA (`checkValue = true`) closes that: the device signs each reading's value, `TelemetryAnchor` counts only readings inside `[minValue, maxValue]` (`inRangeInEpoch`), and `PolicyModule` pays only when at least `minAnchors` readings are both delivered and in range. A device that delivers its full quota out of range is slashed on data quality, not just liveness. Soundness follows from the future-only SLA rule in Section 5: because the range is fixed before the epoch starts, the in-range tally computed while anchoring equals the one settlement recomputes. Tests: `test_Adversarial_OutOfRangeReadingNotPaid`, `test_Adversarial_DeviceCannotForgeInRange`, `testFuzz_ValuePayIffInRangeGteMin`.

---

8. Permissionless keepers

`settleEpoch` is open to anyone. To decentralize the driver without weakening the guarantee, the vault holds a dedicated `keeperPool` and an immutable `keeperReward`. On each settlement the caller receives an equal tip drawn **only** from the keeper pool. Pay and slash amounts are computed and transferred first and are never touched; when the pool is empty the tip is zero and settlement still proceeds. Because the tip is equal on either

verdict, a keeper has no incentive to prefer paying or slashing. The single Settlement Agent thus becomes one of many incentivized, permissionless keepers; a Go settler can run as one keeper of many by hash-partitioning devices and treating `AlreadySettled` as success. Tests: `test_KeeperTipOnPay_OperatorAmountUnchanged`, `test_KeeperTipOnSlash_TreasuryAmountUnchanged`, `test_KeeperEmptyPool_TipZero_SettlementProceeds`.

---

9. Native BOT Chain integration

PulseGrid is not a plain redeploy; it exploits properties specific to BOT Chain.

- **Sub-second blocks (~0.74s measured).** Real-time, per-interval settlement is viable at all.
- **Near-zero fees (base fee 0).** Continuous micro-settlement is economically sane.
- **Gasless for plain wallets.** Devices sign but never transact; a sponsor relayer pays the gas. BOT Chain advertises an EOA paymaster (MegaFuel / BEP-414), but no public paymaster endpoint exists on mainnet or testnet and it does not sponsor type-3 blob transactions, so PulseGrid runs a self-hosted, MegaFuel-shaped sponsor (logged as BOUNTY_LOG B-008). Devices are provably zero-gas either way, and the device path points at a native paymaster the moment one ships.
- **Blob transactions (EIP-4844).** Raw telemetry rides in blobs as the data-availability layer, with a compact on-chain hash as the permanent settlement proof.

---

10. Economics

Settlement is in canonical, bridge-backed **USDT** (6 decimals). The vault is asset-agnostic: the token is a constructor argument, so retargeting another stablecoin is a one-line change at deploy.

Two balances back a device. The funder's **escrow** pays the operator on satisfied intervals; the operator's **bond** is slashed on missed intervals and is what gives the operator skin in the game. Both are per-device and separate; a bond cannot be withdrawn while finalized obligations remain unsettled.

Devices spend zero gas. The sponsor relayer pays roughly **0.04 BOT per anchor** and each settlement costs roughly **0.03 BOT**, at BOT Chain's ~50 gwei gas floor. Because that floor makes always-on streaming non-trivial, PulseGrid runs the streaming fleet and settler in short **funded windows** rather than 24/7; the on-chain settled ledger is permanent regardless. Scaling reduces per-settlement overhead through batching and a native paymaster.

Why no token. Settlement in a real stablecoin is the product; the guarantee is what has value, and it comes from on-chain policy, not from an asset PulseGrid issues. The incentive layer, keeper tips and operator bonds, needs no token. PulseGrid therefore has no native token, and settled amounts are real dollars, not a volatile unit.

---

11. Security and threat model

Each participant is assumed potentially adversarial; the design neutralizes each.

- **Settlement Agent or keeper.** Can only supply `(deviceId, epoch)`; the contract re-derives the verdict. Cannot overpay, wrongly slash, pay an un-anchored interval, or double-settle (Section 6).
- **Operator.** Cannot forge anchors (they require the device signature), cannot pay itself (escrow is the funder's), and cannot escape a pending slash (bond withdrawal is locked until finalized obligations are settled).
- **Funder.** `setSLA` may be called only by the operator or the incumbent funder, so an attacker cannot redirect a device's escrow by naming themselves funder; and settlement uses the SLA that governed the epoch, so terms cannot be repriced retroactively.
- **Sponsor relayer.** Integrity-untrusted: it cannot forge a signature or alter a payload. Withholding is a liveness failure that simply results in a slashable interval.
- **Device.** Replays and stale or future-dated readings are rejected by the nonce and freshness checks; out-of-range readings are slashed by the value layer.
- **Contract-level.** The settled flag is set before every transfer (reentrancy-safe); transfers use a SafeERC20-style path that tolerates non-standard tokens like USDT; and signature recovery rejects malleable (high-`s`) signatures and binds the chain id and verifying contract in the domain, so a signature for one chain or contract cannot be replayed on another.

An external audit and a public bug bounty are planned and are the primary remaining trust items (Section 13); today the guarantee is backed by the adversarial suite, not by a third-party firm.

---

12. Implementation and deployment

PulseGrid is deployed and Blockscout-verified on BOT Chain mainnet (chainId 677), settling in real USDT.

CONTRACT	ADDRESS
DeviceRegistry	`0x3697113dEe694BeEA5FD354384D03ec9198DE7aF`
TelemetryAnchor	`0x6377BC3B08d81e33Fa91927D71CC7f806Fc45662`
ServiceContract	`0x908302Eb8810ef6E40f02dC5Dd63E62bc1cC00Eb`
SettlementVault	`0x6380ebC2EbF72fC6deA07028e500943fb876C546`
USDT (settlement, 6dp)	`0xaBabc7Ddc03e501d190C676BF3d92ef0e6e87a3C`

Deploy block `18822148`. Parameters: `EPOCH_LENGTH` 600s, `maxSkew` 120s, `settlementDelay` 30s, `keeperReward` 0.1 USDT. The contracts are Solidity (Foundry), with **81 passing tests** including the adversarial suite above.

The guarantee is proven live on mainnet: under one value-layer SLA, a device streaming in range is paid and a device streaming out of range is slashed. In-range PAY `0x6b70ff3c…23623a2` (escrow to operator); out-of-range SLASH `0xd326374…f69c75` (bond to treasury). Live totals are read from the app, not frozen here.

---

13. Limitations and future work

- **External audit and bug bounty.** The single largest trust item for real-value settlement; today only an internal adversarial suite exists.
- **Key custody.** Consolidate the deployer, treasury, relayer, and settler roles into a multisig for the treasury and owner, and split the hot keys.
- **Real-hardware pilot.** The browser and CLI paths are on-ramps; a physical device streaming end-to-end is the next credibility step.
- **Native gasless.** Retire the self-hosted sponsor for BOT Chain's paymaster when it ships.
- **Scale.** Batched settlement, richer bonding and reputation curves, and multi-asset settlement via a DEX.

---

14. References

- BOT Chain: scan.botchain.ai, botchain.ai
- EIP-712 (typed structured data), EIP-4844 (blob transactions), EIP-2 (signature malleability).
- BEP-414 / NodeReal MegaFuel (the EOA paymaster PulseGrid targets).

- Source and full architecture: github.com/Manuel-dev01/pulseGrid.

PulseGrid whitepaper v1.0. On-chain policy is the sole settlement authority. This document describes a live system on BOT Chain mainnet; it is not investment advice, and PulseGrid has no native token.