Proof of Trust — a git-native blockchain paid in KL bits

Contents

Proof of Trust: A Git-Native Blockchain Whose Consensus is Paid in KL Bits

Version 0.1

Abstract

We propose a blockchain whose canonical ledger is the `main` branch of an ordinary git repository. Blocks are merge commits and code changes are the commits merged in. Consensus is reached by signed probabilistic votes on competing forks, aggregated under a trust-weighted average and scored by the Kullback–Leibler divergence from the realized outcome. The log score is strictly proper, so honest reporting of one's private belief is the unique trust-maximizing strategy in expectation. The block weight used for fork choice is the same quantity that appears in the trust update, so the chain accumulates literal *bits of information* instead of hashing work. We describe the protocol, derive the KL identity, give a worked numerical example, and discuss properties, limitations, and open problems.

1. Motivation

Blockchain consensus has largely oscillated between two costly strategies:

- **Proof of work (PoW).** Security is paid for with physical energy: miners spend electricity to solve hash-preimage puzzles, and the chain accumulates cumulative work. Work is hard to fake and trivially verifiable, at the price of enormous waste.
- **Proof of stake (PoS).** Security is collateralized with the chain's own asset. Misbehavior is punished by slashing stake. Capital cost replaces electrical cost, but voters are paid to *vote*, not to *be right* about anything resolvable.

Both systems collapse meaningful disagreement into a binary "which block is canonical". But a code-change blockchain has a richer question it could answer: "is this change good?" — which invites *probabilistic* reasoning.

This paper proposes **Proof of Trust (PoT)**, where:

1. Voters cast *probability distributions*, not binary approvals, on the set of candidate blocks at a given height.
2. After `K` confirmations, the realized outcome at that height becomes common knowledge, and votes are scored against it by log score.
3. A voter's trust balance grows by the number of nats of KL divergence by which they beat the trust-weighted market, and shrinks by the bits by which they underperformed.

The result is a consensus mechanism where security costs are paid in *information* rather than energy or capital: a voter who is noisy loses trust exponentially; a voter who is accurate grows. A malicious voter with accurate private belief but adversarial reports also loses — strict properness rules out non-truthful reporting as a stable strategy.

2. Git as ledger

A git repository is already an append-only, cryptographically-hashed, content-addressed DAG. Merge commits are natural block boundaries: each merge takes a linear sequence of commits (the pull request) and folds it into the main branch. A PR review is already, conceptually, a probabilistic judgement.

So we take `refs/heads/main` as the **chain**:

- The root commit of `main` is the **genesis**; its message encodes the `ChainParams` (K, alpha, initial trust allocation, …).
- Every other commit on `main` is a **block** whose first parent is the previous block and whose additional parents are the *code transactions* being merged in. The block's metadata (header, trust transactions, votes, proposer signature) lives in a base64-CBOR `PoT-Block:` trailer at the bottom of the commit message.
- Competing proposals for the next block live off `main` under `refs/pot/candidates/<block_id>`. Signed votes live under `refs/pot/votes/h-<height>/<voter>`.

Every git repo becomes a self-contained chain. Gossip reduces to `git push`/`git fetch` of the `refs/pot/*` namespace.

3. Protocol sketch

3.1 Objects

Three payload types make up a block, in addition to the standard header:

- **Code transactions**: the git commits being merged; opaque to consensus.
- **Trust transactions**: `(from, to, amount, nonce)` signed by `from`, transferring trust between accounts.
- **Votes**: for some earlier height `h`, a probability distribution `q_v` over the set of candidate blocks at `h`, plus a residual "none" slot.

Probabilities are encoded as integers in parts per million summing to `PPM_TOTAL = 1 000 000`, so the cryptographic hash of a signed vote never depends on floating-point subtleties.

3.2 Consensus

Let the tip be at height `t`. Each voter `v` has a non-negative trust balance `t_v`. For any candidate block `B` at height `t + 1`, define its **weight**

$$W(B) \;=\; \sum_{v} t_v^{(t)}\, q_v(B)$$

where `t_v^{(t)}` is the voter's trust snapshot at the end of height `t` (the consensus snapshot). The canonical block at height `t + 1` is the heaviest candidate (ties broken by lowest block id). In the full Nakamoto variant, fork choice picks the path from genesis with maximum cumulative weight; in the v0 reference implementation, we apply the rule greedily height-by-height.

3.3 Finalization

A block `B` at height `h` becomes **final** once a descendant at height `h + K` has been appended to the canonical chain. At that moment:

1. The realized winner `y = canonical[h]` is common knowledge.
2. For every admissible vote `q_v` on height `h` (present in the vote ledger), compute the log score `s_v = log q_v(y)` and the market log score `s_m = log p_m(y)`, where `p_m` is the trust-weighted average distribution (§3.2).
3. Update trust: `t_v ← max(0, t_v + alpha * (s_v - s_m))`, where `alpha` is a genesis-fixed constant measured in millitrust per nat.

4. The role of Kullback–Leibler divergence

All the consensus machinery of §3 is really one identity in disguise.

4.1 Log score equals negative KL

For a categorical outcome `y` with realized delta `_y`, and a forecast distribution `q`,

$$S(q, y) \;=\; \log q(y) \;=\; -H(\delta_y) - D_{\mathrm{KL}}(\delta_y \,\|\, q) \;=\; -D_{\mathrm{KL}}(\delta_y \,\|\, q),$$

because `H( _y) = 0`. The score of a forecast is exactly the negative KL divergence between the truth and the forecast. Better forecasts are literally those closer (in bits) to the truth.

4.2 Market-relative payoff

The PoT trust update for voter `v` on winner `y` is

$$\Delta t_v \;=\; \alpha\,[\, S(q_v, y) - S(p_m, y)\,] \;=\; \alpha\,[\, D_{\mathrm{KL}}(\delta_y \| p_m) - D_{\mathrm{KL}}(\delta_y \| q_v)\,].$$

In words: the voter is *paid*, in units of trust, for the KL-bits by which they beat the market, and *taxed* by the bits by which they underperformed.

4.3 Strict properness implies truthful reporting

Let `p*` be the voter's private belief. Their expected trust update is

$$\mathbb{E}_{y\sim p^*}[\Delta t_v] \;=\; \alpha\,[\, \mathbb{E}_{p^*} \log q_v(y) - \mathbb{E}_{p^*} \log p_m(y)\,].$$

The first term is maximized, over choices of `q_v`, iff `q_v = p*` (Gibbs' inequality), and the second term doesn't depend on `q_v`. Therefore *honest reporting uniquely maximizes expected trust gain*, independent of the market. This is the strict-properness property of the log score and is the core incentive-compatibility result for PoT.

4.4 Block weight as market evidence

Normalize `W(B)` across candidates at height `h`: `p_m(B) = W(B) / \sum_{B'} W(B')`. The trust-weighted entropy across candidates is `H(p_m) = -\sum_B p_m(B) \log p_m(B)`. The *information content* of the market's posterior relative to a uniform prior over `n` candidates is

$$I_h \;=\; D_{\mathrm{KL}}(p_m \,\|\, \mathrm{Unif}) \;=\; \log n - H(p_m).$$

This is the largest amount of evidence the market could have registered about which block is canonical at height `h`. If voters are all unsure, `I_h  0`; if the market is concentrated on a single block, `I_h  \log n`. *The chain accumulates `\sum_h I_h` bits of evidence as it grows.* This is the direct analog of Bitcoin's cumulative work, measured in the dimensionless, noise-free unit of information.

4.5 What about conservation?

The log-score update above is **not** zero-sum. Jensen's inequality gives

$$\mathbb{E}_{y\sim p_m} \sum_v \tfrac{t_v}{\sum t_v} \log q_v(y) \;\le\; \log p_m(y),$$

so the trust-weighted average voter underperforms the market log-score. The gap is a form of Jensen "tax" that slowly deflates the total trust supply when voters hold heterogeneous beliefs. Conceptually this is the *cost of disagreement*: a perfectly consensual electorate loses no trust, while a divided one pays a KL fee for its disagreement. In the Kelly-betting variant (future work), a multiplicative update `t_v ← t_v · (q_v(y)/p_m(y))` is exactly zero-sum and reduces asymptotically to the same KL difference per step.

5. Comparisons

System	Security cost	Finality rule	"Work" unit
Proof of Work	electricity	heaviest cumulative hash	hash preimages / bits of entropy
Proof of Stake	locked capital	majority signature	coin-years
Proof of Trust	misreporting cost	heaviest cumulative trust-weighted vote	**KL bits vs. market**

PoT inherits Nakamoto-style probabilistic finality: finality becomes exponentially more robust as `K` increases, because overturning a `K`-confirmed block requires simultaneously reorganizing `K` heights of trust-weighted votes. Unlike PoS, there is no "nothing-at-stake" issue *in expectation* because voting on multiple forks simultaneously loses a voter KL-bits on whichever fork ends up non-canonical.

6. Worked numerical example

Three voters with endowed trust:

voter	trust (mT)
alice	10 000
bob	5 000
carol	5 000

Two candidate blocks `A`, `B` at height 1; parameters `K = 2`, `= 1000` mT/nat. Votes at height 1 (list = `A`, `B`, `none`):

voter	q(A)	q(B)	q(none)
alice	0.90	0.05	0.05
bob	0.50	0.50	0.00
carol	0.10	0.85	0.05

Block weights

```
W(A) = 10000·0.9 + 5000·0.5 + 5000·0.1 = 9000 + 2500 + 500 = 12 000
W(B) = 10000·0.05 + 5000·0.5 + 5000·0.85 = 500 + 2500 + 4250 = 7 250
```

Fork choice elects `A`. Heights 2 and 3 are appended (any contents), triggering finalization of height 1 at block 3.

Market distribution at height 1

Total voter trust `S = 20 000`. Aggregated candidates `{A, B}`.

```
p_m(A)    = 12 000 / 20 000 = 0.6000
p_m(B)    =  7 250 / 20 000 = 0.3625
p_m(none) =    750 / 20 000 = 0.0375
```

Information content

```
H(p_m) over 3 outcomes = -(0.6 log 0.6 + 0.3625 log 0.3625 + 0.0375 log 0.0375)
                       0.8213 nats
log 3                  1.0986 nats
I_1 = D_KL(p_m || Unif) = log 3 - H(p_m)
                       0.2773 nats    0.400 bits
```

Height 1 contributes about 0.4 bits of evidence to the chain.

Trust updates

Winner `y = A`. `p_m(y) = 0.6`. `log 0.6  -0.5108`.

voter	q_v(A)	log q_v(A)	margin (nats)	Δ (mT)
alice	0.9	-0.1054	+0.4055	**+405**
bob	0.5	-0.6931	-0.1823	**-182**
carol	0.1	-2.3026	-1.7918	**-1792**

New balances: alice 10 405, bob 4 818, carol 3 208. Total 18 431 mT; the system lost 1 569 mT to the Jensen tax because voter beliefs were far from the market average.

The integration test in `crates/pot-git/tests/e2e.rs` runs this exact scenario end-to-end on a real git repository and asserts these numbers to the millitrust.

7. Properties

- **Strict properness** (§4.3): honest reporting uniquely maximizes expected trust gain.
- **Sybil resistance**: trust has to come from somewhere. New identities gain trust only via transfer or by outperforming the market. Creating N shell voters who all copy the market gets no trust (their margin vs. market is 0).
- **Information-monotone finality**: each finalized height increases `\sum I_h`, and a reorg at depth `K` requires the attacker to at least match the evidence content the attacked chain has accumulated.
- **Deterministic replay**: the full state is a pure function of `canonical` and `ChainParams`. This is what allows `pot status` and `pot finalize` to work by reading the git repo alone.
- **Transparent audit**: every finalized scoring event appears in a `ScoringReport` that any observer can reconstruct from the canonical chain.

8. Limitations and open problems

1. **Cold start.** Genesis must seed some trust to some set of pubkeys. That initial allocation is politically loaded.
2. **Collusion**. Rational voters with correlated beliefs are correctly rewarded together; *adversarially* colluding voters who vote identically hurt the market's accuracy and drag everyone's trust down. A quorum-based collusion attack can fix a fork for short-term gain if the colluders collectively hold more than the honest trust.
3. **Oracle manipulation of `K`.** `K` is a genesis constant here. Governance-upgradable `K` is left as future work.
4. **Nothing-at-stake analog.** A voter voting identically on both sides of a fork is not free: on whichever side ends up non-canonical they lose exactly their KL to the truth. But they can still *minimize* loss by voting uniform, which is suboptimal but only marginally punitive. Stronger slashing for double-voting is future work.
5. **Conservation.** The log-score update is not exactly zero-sum (§4.5). A multiplicative Kelly-like update is; extending PoT in that direction is future work.
6. **Floating-point reproducibility.** The reference implementation uses IEEE 754 `f64` for scoring. On heterogeneous hardware this is generally deterministic but not guaranteed; a full production implementation would use a canonical fixed-point log or lookup-table approach over `probs_ppm`.

9. Client ecosystem

The reference implementation ships a family of single-purpose clients against the same git repository:

- `pot` — umbrella CLI (init / keygen / propose / advance / finalize).
- `pot-bootstrap` — P2P gossip daemon speaking a custom length-prefixed CBOR protocol over TCP; relays signed votes, trust-transfer transactions, and prediction-market objects.
- `pot-light` — headers-only verifier that trusts only proposer signatures and the first-parent chain, suitable for constrained devices and auditors.
- `pot-market` — prediction-market client that uses the v0.2 protocol extension to host arbitrary questions (not just block canonicality) with the same strictly-proper KL-bit scoring rule.
- `pot-web` — browser UI and JSON API over the chain, with optional local signing endpoints.
- `pot-deploy` — canonical-tree runner: checks out the tip's code commits into a detached git worktree and executes declared entrypoints on each tip advance and on each finalization.

A fuller per-client walkthrough lives in `clients.md`; the wire protocol and light-client rules are normative in §§11–12 of `spec.md`.

10. Conclusion

Proof of Trust replaces proof-of-work hashing and proof-of-stake locking with something cheaper and more meaningful: *information*. Kullback–Leibler divergence is not a bolted-on metric but the unifying quantity that governs block weight, trust updates, incentive compatibility, and cumulative finality. Anchoring the whole thing on a git repository makes the resulting system a natural fit for the one domain where probabilistic consensus has always been implicit: code review.

Appendix A. Agent value in prediction markets

Merged from the former `market_math.md` *at repo root — mechanism-agnostic KL scoring used by aion-core markets and PoT trust updates.*

Abstract. We present a mechanism-agnostic framework for quantifying agent value in prediction markets based on Kullback-Leibler divergence reduction, adjusted for prediction timeliness and market-specific cost structures. The system value function $V_{\text{system}}(a)$ aggregates agent contributions across markets, incorporating temporal decay and mechanism-dependent normalization factors.

1. Preliminaries. Let $\mathcal{M} = \{1, \dots, M\}$ denote markets and $\mathcal{K} = \{1, \dots, K\}$ agents. For market m, outcome space $\mathcal{L}_m$ and final distribution P_m^{final} at $t_1^{(m)}$. Bets $q_{m,t} \in \Delta(\mathcal{L}_m)$ with ϕ_m mapping time to agents.

2. Core value. The information value of bet t in market m:

$$v_{m,t} = D_{\text{KL}}(P_m^{\text{final}} \| P_{m,t-1}) - D_{\text{KL}}(P_m^{\text{final}} \| P_{m,t})$$

Temporal decay: $\delta(t) = \exp(-\lambda(t_1 - t))$ or $\delta(t) = 1 - t/t_1$. Adjusted value: $v_{m,t}^{\text{adj}} = \delta(t) \cdot v_{m,t}/c_{m,t}$.

3. Mechanism costs. Bayesian: $c_{m,t}^{\text{Bayes}} = \|\alpha_{t-1}\|_1$. LMSR: $c_{m,t}^{\text{LMSR}} = \nabla C(q_{t-1}) \cdot b_t$.

4. System value.

$$V_{\text{system}}(a) = \sum_{m,t:\phi_m(t)=a} \delta(t) \cdot \frac{D_{\text{KL}}(P_m^{\text{final}} \| P_{m,t-1}) - D_{\text{KL}}(P_m^{\text{final}} \| P_{m,t})}{c_{m,t}}$$

Value is additive across agents; $V_{\text{system}}(a)$ equals counterfactual KL without agent a.

5. Incentive alignment. Alignment gap $\Gamma(a) = V_{\text{system}}(a) - \mathbb{E}[R(a)]$ depends on mechanism (Bayesian scoring rule vs LMSR profit).

6. Conclusion. $V_{\text{system}}(a)$ is the shared primitive behind aion-core prediction markets and PoT trust growth when forecasts are scored in bits.