LVS — Drift-Based Consensus Specification (EN)

Version 1.0 — Final, Mathematical & Algorithmic Specification

---

1. Purpose of This Document

This specification defines the full mathematical and algorithmic foundation of **Drift-Based Consensus (DBC)** — the core mechanism that allows LVS to reach global state convergence without: - miners, - validators, - committees, - block production, - identity, - voting, - staking.

DBC provides a fully autonomous, emergent, self-regulating consensus layer.

This document is technical and formal. It serves as the foundation for all implementations and academic evaluation.

---

2. Consensus Philosophy

Classical consensus requires *agreement on discrete events* (blocks, votes, signatures).
LVS replaces this with *continuous state convergence* using drift dynamics.

Consensus emerges not through authority — but through **probabilistic correction forces** distributed across micro-nodes.

DBC is: - continuous, - local-first, - correction-driven, - identity-free, - authority-free.

---

3. Mathematical Model Overview

Let: - `S` be the global system state - `Si` be the partial local state at node i - `VU` represent Value Units - `TC` represent Trust Credits - `EP` be entropy packet input - `DC` be drift coefficients

The goal:

```
lim (t → ∞) Si(t) = S*
```

Where **S*** is the globally converged equilibrium.

DBC guarantees convergence *without* synchronized clocks, deterministic ordering, or block production.

---

4. Core Drift Equation

Each node updates its partial state via:

```
Si(new) = Si(old) + D
```

Where D (drift vector) is:

```
D = α * F(EP, Si, DC) + β * G(SDM_buffer)
```

Meaning: - **F()** applies drift from entropy packet - **G()** integrates state diffs from peers - `α` and `β` are weighting constants controlled by the network

4.1 α (alpha) — Entropy Drift Weight

Controls how strongly new entropy affects the node. Typical range: `0.01 – 0.10`.

4.2 β (beta) — Peer Correction Weight

Controls how strongly peer diffs affect convergence. Typical range: `0.05 – 0.20`.

---

5. Entropy Packet (EP) Influence

EP is:

```
EP = { entropy_vector E, timestamp T, node_load L }
```

We define the entropy-induced drift as:

```
F(EP) = normalize(E) * drift_intensity(L)
```

Where: - `normalize(E)` scales entropy to stable boundaries - `drift_intensity(L)` reduces impact if node is overloaded

This introduces controlled randomness — the engine of emergent consensus.

---

6. State Diff Integration

Nodes store an SDM buffer of recent diffs from peers.

Each SDM:

```
SDM = { shard_id, diff_vector dv, drift_weight w }
```

Correction contribution:

```
G(SDM_buffer) = Σ ( dv * w )
```

Weighted sum ensures: - high-quality diffs dominate - malicious noise is suppressed

---

7. Drift Coefficients (DC)

DC provide local correction scaling:

```
DC = f(VU, TC, entropy_variation)
```

7.1 VU Effect

Nodes with high VU represent active contribution:

```
weight_VU = log(1 + VU)
```

7.2 TC Effect

Nodes with high TC are more stable:

```
weight_TC = sqrt(TC)
```

7.3 Entropy Variation

Large entropy swings reduce trust temporarily:

```
entropy_dampen = 1 / (1 + variance(E))
```

Total coefficient:

```
DC_total = weight_VU * weight_TC * entropy_dampen
```

---

8. Global Convergence Guarantees

Convergence emerges because:

1. Nodes constantly apply drift corrections.
2. Redundancy ensures shared state fragments.
3. Diffs propagate correction forces across the network.
4. Entropy prevents stagnation in local minima.
5. VaultGuard prevents runaway negative drift.

In the limit, the system reaches stable equilibrium.

---

9. VaultGuard Integration

VaultGuard enforces global safety constraints.

9.1 Hard Invariant

```
VU >= 0
```

If drift attempts to push VU negative:

```
VU = 0
stop_drift_for_cycle()
```

9.2 Anomaly Detection

If diff magnitude exceeds threshold:

```
|dv| > anomaly_limit
```

Node marks source as unstable and down-weights future diffs.

9.3 Recovery Trigger

If state imbalance > force_limit:

```
activate_recovery_mode()
```

recovery_mode reduces α, increases β — prioritizing convergence.

---

10. Consensus Cycle Algorithm (Pseudocode)

```
loop:
    EP = receive_entropy()
    SDMs = receive_diffs()

    D1 = alpha * F(EP)
    D2 = beta  * G(SDMs)

    D = D1 + D2

    if violates_vaultguard(D):
        D = vaultguard_correct(D)

    Si = Si + D

    dv = compute_local_diff()
    broadcast_SDM(dv)
end loop
```

---

11. Attack Resistance

11.1 51% Attacks Impossible

No validators → no majority.

11.2 Sybil Attacks Mitigated

Fake nodes cannot: - override convergence, - create blocks, - vote, - inject identities.

Because identities do not exist.

11.3 Noise Attacks Limited

Entropy is expected → drift stabilizes noise.

11.4 Drain Attacks Blocked

VaultGuard clamps destructive diffs.

---

12. Convergence Proof (Informal)

DBC resembles a **stochastic iterative solver**:

```
Si(t+1) = Si(t) + α*F + β*G
```

This is a contraction mapping under bounded drift values:

```
|Si(t+1) - S*| < k * |Si(t) - S*|
```

Where 0 < k < 1.

Thus, convergence is guaranteed.

---

13. Performance Characteristics

- **CPU usage:** minimal
- **Packet loss:** tolerated
- **Latency:** tolerated
- **Nodes:** infinite horizontal scalability

DBC has no bottlenecks and no global synchronization.

---

14. Implementation Requirements

A compliant node MUST: - implement EP and SDM formats - follow drift update formula - enforce VaultGuard invariants - support shard redundancy - maintain correction buffers

---

15. Conclusion

DBC is the first consensus model designed to operate: - without identity, - without leaders, - without blocks, - without voting, - without trust in participants.

It ensures that LVS remains autonomous, self-balancing, and resilient across any global condition.

This specification is final and stable, forming the backbone of all node implementations.