LVS — MVP / Prototype Architecture (EN)

Version 1.0 — Final, Developer-Focused Implementation Guide

---

1. Purpose of This Document

This document defines the **minimum viable architecture** required to build the **first working prototype (MVP)** of the LVS Autonomous Value Layer.

Its goal is simple: - turn the theory into a running system, - with minimal complexity, - while keeping all core LVS principles intact.

This is the document developers will use to **start writing code immediately**.

---

2. MVP Scope

The MVP must demonstrate: 1. **Micro-node communication** (browser ↔ browser or browser ↔ server) 2. **Entropy packet exchange (EP)** 3. **Drift cycle execution (DBC-lite)** 4. **State diffs (SDM) propagation** 5. **Local convergence behavior** between nodes

It does *not* need: - full VaultGuard implementation, - full sharding, - optimized consensus, - full redundancy logic.

We build a minimal working version of the LVS engine.

---

3. MVP Architecture Diagram (Text Form)

```
+---------------------+
|  Browser Node A     |
|  - Entropy Engine   |
|  - Drift Core       |
|  - Local State      |
|  - P2P Transport    |
+---------------------+
          ↑    ↓
   WebRTC / WebSocket
          ↑    ↓
+---------------------+
|  Browser Node B     |
|  - Entropy Engine   |
|  - Drift Core       |
|  - Local State      |
|  - P2P Transport    |
+---------------------+
```

Optional:

```
+----------------------+
|  Microserver Node    |
|  (Go/Rust)           |
|  - Relay / Peer Hub  |
|  - Storage Cache     |
+----------------------+
```

The MVP requires only **two browser nodes** to show drift convergence.

---

4. Components Required for MVP

We reduce the full LVS architecture into three essential modules.

4.1 Transport Module (MVP)

Goal:

Allow nodes to exchange: - entropy packets (EP), - state diffs (SDM).

MVP Choice: WebRTC DataChannel (browser)

```
openConnection(peer)
sendMessage(msg)
onMessage(callback)
```

Simple JSON message format.

4.2 Entropy Engine (MVP)

Minimal entropy vector:

```
E = [ random(), random() ]
```

Only 2D or 3D vector needed.

MVP function:

```
function generateEntropy() {
    return [Math.random(), Math.random()];
}
```

4.3 Local State (MVP)

We simplify LVS state to:

```
state = {
    vu: number,   // value-like scalar
    tc: number,   // trust-like scalar
    drift: [x,y]  // drift offset vector
}
```

4.4 Drift Core (MVP)

Drift = entropy influence + peer influence

```
D = alpha * normalize(E) + beta * avg(peerDiffs)
```

MVP values:

```
alpha = 0.05
beta  = 0.10
```

Update:

```
state.drift[0] += D[0]
state.drift[1] += D[1]
```

---

5. State Diff Message (MVP)

Minimal SDM:

```
{
  type: "sdm",
  diff: [dx, dy]
}
```

Generated after each drift step.

---

6. Drift Cycle (MVP)

Below is the **exact MVP pseudocode**.

```
function driftCycle() {
    // 1. entropy
    let E = generateEntropy();

    // 2. peer messages
    let diffs = receivedDiffs(); // array of [dx,dy]
    let peerInfluence = average(diffs);

    // 3. drift calculation
    let D = [
        alpha * E[0] + beta * peerInfluence[0],
        alpha * E[1] + beta * peerInfluence[1]
    ];

    // 4. apply drift
    state.drift[0] += D[0];
    state.drift[1] += D[1];

    // 5. broadcast diff
    broadcast({type:"sdm", diff: D});
}
```

This loop runs every **50–200 ms**.

---

7. MVP Convergence Behavior

Two nodes starting with different drift vectors:

```
Node A: [5, 1]
Node B: [-2, -3]
```

After 20–40 cycles: - both nodes drift toward a shared equilibrium, - values move closer on every iteration, - network self-stabilizes.

This **visual convergence** is the core demonstration of LVS.

8. MVP User Interface (Optional)

A minimal UI is recommended: - show each node as a dot on 2D plane, - color indicates entropy intensity, - connecting line shows P2P link, - drift vector moves the dot gradually.

This makes LVS visually demonstrable.

9. MVP Success Criteria

The MVP is considered successful if it demonstrates:

1. Node-to-node communication

Two browser nodes successfully exchange entropy and diffs.

2. Drift cycles execute

State updates smoothly.

3. Convergence appears visually

Nodes move toward similar drift values.

4. System self-stabilizes

No collapse, no runaway behavior.

10. MVP Tech Stack Recommendations

Browser Prototype:

- TypeScript
- WebRTC DataChannels
- Simple Canvas/WebGL visualization

Server Relay (optional):

- Node.js / Go
- WebSocket signaling for WebRTC

Hosting:

- github.io or lvs.network

11. Next Steps After MVP

After MVP validation: 1. Add VaultGuard Lite 2. Add basic redundancy (duplicate state copies) 3. Add minimal sharding 4. Expand to 20–100 nodes 5. Release public demo

---

12. Conclusion

This MVP / Prototype Architecture translates LVS theory into a concrete, minimal, working system.
It defines everything required for developers to start coding immediately, while maintaining all essential principles of LVS.

This is the first real step toward a running Testnet.