

# A Formal Scaling Argument for DSM in Adversarial Real-World Networks

Execution-Plane and Storage-Plane Scalability Without Global Coupling

Brandon “Cryptskii” Ramsay

February 7, 2026

## Abstract

We give a formal, end-to-end scaling argument for DSM (Deterministic State Machine) under real-world conditions: arbitrary latency, packet loss, reordering, partitions, retries, and device pauses. The model is deterministic and avoids wall-clock time by representing the environment as an adversarial asynchronous scheduler over discrete delivery steps. DSM scales in two planes: (i) an execution plane of relationship-local state machines, and (ii) a storage plane of deterministic replication over capacity-limited storage nodes. We prove safety under adversarial scheduling, non-interference (unrelated activity cannot invalidate a device’s witnesses), additive execution scaling (matchings in the relationship graph advance in parallel), and scalable online availability via deterministic replica placement, mirroring, repair, and quorum fetch. We further model explicit per-node capacity limits and show deterministic registry expansion increases total capacity without inducing global coupling.

## Contents

<b>1</b>	<b>Thesis</b>	<b>2</b>
<b>2</b>	<b>Design goals and non-goals</b>	<b>2</b>
2.1	Goals . . . . .	2
2.2	Non-goals . . . . .	2
<b>3</b>	<b>Part I: Execution-plane formal model</b>	<b>3</b>
3.1	Devices and relationship graph . . . . .	3
3.2	Canonical encoding and hash adjacency . . . . .	3
3.3	Per-device roots commit only to incident edges . . . . .	3
3.4	Acceptance predicate . . . . .	3
3.5	Fork exclusion (tripwire) . . . . .	4
<b>4</b>	<b>Part I-A: Real-world network model (no wall-clock)</b>	<b>4</b>
<b>5</b>	<b>Part I-B: Execution-plane theorems</b>	<b>4</b>
<b>6</b>	<b>Part II: Storage-plane model (online availability without all-in-all replication)</b>	<b>5</b>
6.1	Why storage exists and what it stores . . . . .	5
6.2	Registry and deterministic placement . . . . .	5
6.3	Publish, mirror, fetch, quorum . . . . .	6

6.4	Per-node capacity and admission . . . . .	6
6.5	Node commitments: NodeSMT and ByteCommit chain . . . . .	6
6.6	Mirroring and repair under churn . . . . .	6
<b>7</b>	<b>Part II-A: Registry evolution (expansion without voting)</b>	<b>7</b>
<b>8</b>	<b>Part II-B: Storage-plane theorems</b>	<b>7</b>
<b>9</b>	<b>Part III: Coupling the planes (online transactions without global ordering)</b>	<b>8</b>
<b>10</b>	<b>Part IV: Resource model (real networks without time)</b>	<b>8</b>
<b>11</b>	<b>Part V: Load model (why “not all-in-all” stays balanced)</b>	<b>8</b>
<b>12</b>	<b>Threat model and assumptions</b>	<b>9</b>
<b>13</b>	<b>Conclusion</b>	<b>9</b>
<b>A</b>	<b>Appendix A: Minimal invariants checklist</b>	<b>9</b>

## 1 Thesis

DSM scales because it deletes the global serialized object in *both* planes:

- **Execution plane:** no global ordering, no global state root, no global witness refresh.
- **Storage plane:** no all-in-all replication, no global fetch requirement, no single availability stream.

The environment can slow particular edges or replica sets, but cannot create a universal bottleneck because acceptance and verification remain local and deterministic.

## 2 Design goals and non-goals

### 2.1 Goals

1. **Determinism:** identical inputs imply identical acceptance decisions.
2. **No global coupling:** unrelated activity cannot invalidate your local proofs/witnesses.
3. **Online and offline viability:** offline bilateral progress is possible; online unilateral delivery is supported via storage nodes.
4. **Capacity transparency:** each storage node has explicit limits, signaled deterministically, enabling registry expansion.

### 2.2 Non-goals

This paper does not define a specific cryptosystem for endpoint authorization (signatures/commitments are abstracted), nor does it specify implementation-level transport details (BLE, TCP, etc.). Those are separable from the scaling law.

### 3 Part I: Execution-plane formal model

#### 3.1 Devices and relationship graph

**Definition 1** (Devices and relationships). Let  $\text{Dev}$  be a set of devices (or identities). A relationship set is

$$\text{Rel} \subseteq \{\{i, j\} : i, j \in \text{Dev}, i \neq j\}.$$

Define the relationship graph  $G = (\text{Dev}, \text{Rel})$ .

**Definition 2** (Incident sets). For  $u \in \text{Dev}$  define

$$\text{Rel}(u) = \{\{u, v\} \in \text{Rel}\}, \quad \text{Chains}(u) = \{C_{u,v} : \{u, v\} \in \text{Rel}\}.$$

#### 3.2 Canonical encoding and hash adjacency

**Assumption 1** (Canonical encoding). Every transition payload  $\text{Tx}$  has a unique canonical byte encoding  $\text{enc}(\text{Tx}) \in \{0, 1\}^*$ .

**Assumption 2** (Collision resistance).  $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$  is collision-resistant.

**Definition 3** (Relationship-local chains). For each  $\{i, j\} \in \text{Rel}$  there exists a forward-only chain

$$C_{i,j} = (s_{i,j}^{(0)}, s_{i,j}^{(1)}, \dots), \quad s_{i,j}^{(t)} \in \{0, 1\}^\lambda,$$

with adjacency rule

$$s_{i,j}^{(t+1)} = H(s_{i,j}^{(t)} \parallel \text{enc}(\text{Tx}_{i,j}^{(t+1)})). \quad (1)$$

#### 3.3 Per-device roots commit only to incident edges

**Assumption 3** (Deterministic map commitment). There exists a deterministic commitment  $\text{Commit}$  over finite maps  $M : \text{Dev} \rightarrow \{0, 1\}^\lambda$  with deterministic membership proofs.

**Definition 4** (Device map and root). Each device  $u$  maintains a map  $M_u$  with

$$\text{dom}(M_u) = \{v : \{u, v\} \in \text{Rel}\}, \quad M_u[v] = \text{tip}(C_{u,v}) \in \{0, 1\}^\lambda.$$

The device root is  $r_u = \text{Commit}(M_u)$ .

**Definition 5** (Witness). A witness  $\text{Wit}_u(u, v)$  is a membership proof that  $M_u[v] = s$  under root  $r_u$ .

#### 3.4 Acceptance predicate

**Definition 6** (Acceptance predicate). A proposed update on  $\{i, j\}$  with payload  $\text{Tx}$  is acceptable iff

$$\text{Acc}(i, j, \text{Tx}, \text{Wit}_i, \text{Wit}_j) = 1,$$

where acceptance checks only:

1. **Adjacency:**  $s' = H(s \parallel \text{enc}(\text{Tx}))$  for prior tip  $s$ .
2. **Authorization:**  $\text{Tx}$  contains required endpoint authorizations by  $i$  and  $j$ .
3. **Membership:**  $\text{Wit}_i$  proves  $M_i[j] = s$  under  $r_i$ , and  $\text{Wit}_j$  proves  $M_j[i] = s$  under  $r_j$ .
4. **Local update:** only  $M_i[j]$  and  $M_j[i]$  change from  $s$  to  $s'$ .
5. **Fork exclusion:** an honest verifier will not accept two distinct successors from the same predecessor on the same chain.

### 3.5 Fork exclusion (tripwire)

**Definition 7** (Fork). A fork on  $C_{i,j}$  at predecessor  $s$  is a pair of distinct successors

$$x = H(s \parallel \text{enc}(\text{Tx}_x)), \quad y = H(s \parallel \text{enc}(\text{Tx}_y)), \quad \text{enc}(\text{Tx}_x) \neq \text{enc}(\text{Tx}_y).$$

## 4 Part I-A: Real-world network model (no wall-clock)

**Definition 8** (Messages). Let  $\text{Msg}$  be the set of protocol messages. Each  $m \in \text{Msg}$  is a finite string in  $\{0, 1\}^*$ .

**Definition 9** (Delivery steps and adversarial scheduling). Execution proceeds in discrete delivery steps  $k = 0, 1, 2, \dots$ . At step  $k$ , devices may attempt sends

$$\text{Send}_k \subseteq \text{Dev} \times \text{Dev} \times \text{Msg},$$

and the adversarial scheduler chooses deliveries

$$\text{Deliver}_k \subseteq \text{Dev} \times \text{Dev} \times \text{Msg}.$$

The scheduler may delay, reorder, duplicate, or drop arbitrarily. “Latency” is modeled as many steps of nondelivery.

**Assumption 4** (Eventual delivery (fairness)). If device  $i$  continues attempting to send message  $m$  to reachable device  $j$ , then  $m$  is delivered at some future step. No bound is assumed.

## 5 Part I-B: Execution-plane theorems

**Lemma 1** (Non-interference). *Fix  $u \in \text{Dev}$ . Any sequence of accepted transitions  $\pi$  that updates only relationships in  $\text{Rel} \setminus \text{Rel}(u)$  leaves  $r_u$  unchanged, and all membership witnesses for chains in  $\text{Chains}(u)$  remain valid after  $\pi$ .*

*Proof.* A transition on  $\{i, j\}$  updates only  $M_i[j]$  and  $M_j[i]$ . If  $\{i, j\} \notin \text{Rel}(u)$  then  $u \notin \{i, j\}$ , so  $M_u$  is untouched throughout  $\pi$ . Thus  $r_u = \text{Commit}(M_u)$  is unchanged, and membership proofs into an unchanged deterministic commitment remain valid.  $\square$

**Theorem 1** (Safety under adversarial scheduling). *Under collision resistance and canonical encoding, for any scheduler  $\text{Sched}$ , honest devices never accept an invalid relationship transition.*

*Proof.* Acceptance is a pure predicate over adjacency (1), authorization, and membership proofs into current roots. Scheduling affects when a candidate is seen, not whether  $\text{Acc}$  holds. Invalid candidates fail  $\text{Acc}$  and are rejected deterministically. Fork exclusion prevents accepting conflicting successors.  $\square$

**Theorem 2** (Relationship liveness under eventual delivery). *Assume eventual delivery. If endpoints  $i$  and  $j$  continue attempting to advance  $C_{i,j}$  with valid transitions, then  $C_{i,j}$  advances.*

*Proof.* By eventual delivery, the required proposal/proof messages arrive in some future steps. Upon delivery they satisfy  $\text{Acc}$  by construction, so both endpoints accept.  $\square$

**Definition 10** (Conflict and matching). Two relationship transitions conflict iff they share an endpoint. A matching  $M \subseteq \text{Rel}$  is a set of relationships where no device appears in more than one edge.

**Theorem 3** (Parallel acceptability on matchings). *For any matching  $M$ , DSM can accept one valid transition on every relationship in  $M$  in parallel. This independence holds under any scheduler Sched.*

*Proof.* Acceptance on  $\{i, j\}$  references only  $i, j$  and  $C_{i,j}$ . In a matching, no device participates twice, so no root/map is written by two transitions. Thus checks and updates commute across edges. Scheduling may delay some edges, but cannot create cross-edge dependencies because no global object is referenced.  $\square$

**Corollary 1** (Additive execution scaling). *Whenever the environment delivers the necessary messages for edges in a matching  $M$ , DSM can advance at least  $|M|$  relationships concurrently.*

## 6 Part II: Storage-plane model (online availability without all-in-all replication)

### 6.1 Why storage exists and what it stores

**Definition 11** (Object taxonomy). Let  $\text{Obj} \subseteq \{0, 1\}^*$  be the set of storable objects. Objects are partitioned into classes:

1. **Transition blobs** (payloads referenced by relationship chains),
2. **Receipts/proofs** (witness artifacts needed for later verification),
3. **Advertisements** (discoverability objects, e.g. vault ads),
4. **Commitments** (node commitments such as ByteCommit, see below).

**Definition 12** (Addressing and integrity). Each object  $o \in \text{Obj}$  has address  $\text{Addr}(o) = \text{H}(o) \in \{0, 1\}^\lambda$  and size  $\text{size}(o) \in \mathbb{N}$ . A client accepts a fetched object for address  $a$  only if  $\text{H}(o) = a$ .

This makes storage nodes non-authoritative: they serve bytes; truth is verified client-side.

### 6.2 Registry and deterministic placement

**Definition 13** (Storage nodes and registry). Let  $\text{Node}$  be a set of storage nodes. The active registry at step  $k$  is  $\text{Reg}_k \subseteq \text{Node}$  together with any metadata needed for placement and transport.

**Assumption 5** (Deterministic permutation). There exists a deterministic function  $\text{perm}$  producing a permutation of active nodes from a seed:

$$\text{perm}(\text{Reg}_k, \text{key}) \rightarrow (n_1, \dots, n_{|\text{Reg}_k|}),$$

where  $\text{key} \in \{0, 1\}^\lambda$ .

**Definition 14** (Deterministic replica placement). Fix replication factor  $r \geq 1$ . Define placement as:

$$\text{Place}(\text{Reg}_k, a, r) = (n_1, \dots, n_r) \quad \text{where } (n_1, \dots) = \text{perm}(\text{Reg}_k, a).$$

Define replica set  $\text{Rep}_k(a) = \{n_1, \dots, n_r\}$ .

This is the formal “not all-in-all”: an object is assigned to exactly  $r$  nodes in  $\text{Reg}_k$ .

### 6.3 Publish, mirror, fetch, quorum

**Definition 15** (Publish attempt). To publish object  $o$  with address  $a = \text{Addr}(o)$  at step  $k$ , the publisher sends  $o$  to each  $n \in \text{Rep}_k(a)$ . A node may reply with an acknowledgment  $\text{ack}(n, a) \in \{0, 1\}^*$ .

**Definition 16** (Quorum fetch). Fix a quorum threshold  $q \leq r$ . A client treats  $a$  as *available* at step  $k$  if it can fetch objects  $o_1, \dots, o_q$  from distinct nodes in  $\text{Rep}_k(a)$  such that

$$\forall t : H(o_t) = a \quad \text{and} \quad o_1 = \dots = o_q.$$

**Claim 1** (Non-authoritative storage). *No single node can lie the client into accepting incorrect bytes for  $a$  because acceptance requires  $H(o) = a$ . Quorum fetch provides redundancy under failure/partition.*

### 6.4 Per-node capacity and admission

**Definition 17** (Capacity and utilization). Each node  $n$  has capacity  $\mathfrak{m}(n) \in \mathbb{N}$  bytes and used space  $\text{used}_k(n) \in \mathbb{N}$  at step  $k$ . Utilization is  $\text{util}_k(n) = \text{used}_k(n) / \mathfrak{m}(n)$ .

**Definition 18** (Admission predicate). A node admits an object  $o$  at step  $k$  iff

$$\text{admit}_k(n, o) = 1 \quad \iff \quad \text{used}_k(n) + \text{size}(o) \leq \mathfrak{m}(n).$$

If admitted, the node stores  $o$  and updates  $\text{used}_{k+1}(n)$  accordingly.

This answers your “is there a limit per storage node?” with a formal yes.

### 6.5 Node commitments: NodeSMT and ByteCommit chain

**Assumption 6** (Deterministic node state commitment). Each node maintains a deterministic commitment  $\text{SMT}_k(n) \in \{0, 1\}^\lambda$  to the set of addresses it stores at step  $k$  (e.g. an SMT keyed by addresses with value = size or presence bit).

**Definition 19** (ByteCommit chain). Each node publishes a forward-only commitment

$$\text{ByteCommit}_k(n) = H\left(\text{SMT}_k(n) \parallel \text{enc}(\text{used}_k(n)) \parallel \text{ByteCommit}_{k-1}(n)\right).$$

Clients can use this to audit that a node is consistently reporting its stored set and utilization, without time.

### 6.6 Mirroring and repair under churn

Replica sets can lose members (node failure, removal, partition). Repair restores replication to  $r$ .

**Definition 20** (Replica deficit). For address  $a$  at step  $k$ , define the live replica count as

$$L_k(a) = \left| \{n \in \text{Rep}_k(a) : n \text{ can serve } a\} \right|.$$

A deficit occurs when  $L_k(a) < q$  (availability risk) or  $L_k(a) < r$  (replication below target).

**Definition 21** (Deterministic repair target). Let  $\text{Rep}'_k(a)$  be the replica set computed from the current registry  $\text{Reg}_k$  (placement is deterministic in  $k$ ). Repair means ensuring objects for  $a$  exist on the nodes of  $\text{Rep}'_k(a)$ .

**Assumption 7** (Repair procedure). If an object  $o$  exists on at least one node that can serve it, then a repair agent (any client or node) can copy  $o$  to missing nodes in  $\text{Rep}'_k(a)$ , subject to admission admit.

The repair agent is non-authoritative: it only copies bytes whose address matches.

## 7 Part II-A: Registry evolution (expansion without voting)

**Definition 22** (Capacity signal). Each node emits a deterministic signal derived from its ByteCommit history (abstractly):

$$\text{signal}_k(n) \in \{\text{Up}, \text{Down}, \text{Stable}\}.$$

Intuition: Up when persistently high utilization, Down when persistently low utilization.

**Assumption 8** (Deterministic registry transition). There exists a deterministic transition function

$$\text{next}\left(\text{Reg}_k, \{\text{signal}_k(n)\}_{n \in \text{Reg}_k}\right) \rightarrow \text{Reg}_{k+1},$$

which adds/removes nodes (or partitions) based on signals. No voting, auctions, or wall-clock triggers are required in the model.

## 8 Part II-B: Storage-plane theorems

**Theorem 4** (Bounded-query availability). *Fix  $r$  and  $q \leq r$ . For any address  $a$ , availability verification requires contacting only nodes in  $\text{Rep}_k(a)$  (size  $r$ ). No client must query all nodes, and no node must store all objects.*

*Proof.* By construction,  $\text{Rep}_k(a)$  contains exactly  $r$  nodes. Quorum acceptance is defined solely over distinct nodes in  $\text{Rep}_k(a)$ . Nodes outside  $\text{Rep}_k(a)$  are irrelevant to  $a$ .  $\square$

**Theorem 5** (Per-node capacity does not create global coupling). *Per-node capacities bound how many bytes any individual node can store. They do not induce a universal bottleneck because objects are placed onto bounded replica sets, and total capacity*

$$\mathfrak{m}(\text{Reg}_k) = \sum_{n \in \text{Reg}_k} \mathfrak{m}(n)$$

*grows additively with registry expansion via next.*

*Proof.* No node is required to store all objects; placement assigns each object to  $r$  nodes. If utilization signals cause new nodes to be added,  $\mathfrak{m}(\text{Reg}_k)$  increases by the added capacities. Thus the storage plane scales by adding nodes/partitions, without requiring any global availability stream.  $\square$

**Theorem 6** (Repair preserves target replication under eventual delivery). *Assume eventual delivery and that for each address  $a$  there exists at least one node that can serve  $a$  (or a client that holds  $a$  locally). Then repeated repair attempts restore objects onto the current placement set  $\text{Rep}_k(a)$  subject to admission constraints.*

*Proof.* Repair consists of copying bytes  $o$  whose hash equals  $a$  to missing nodes in  $\text{Rep}_k(a)$ . Under eventual delivery, repair messages arrive. Nodes admit if capacity allows. Repetition plus eventual delivery ensures missing placements are eventually filled whenever admission allows.  $\square$

## 9 Part III: Coupling the planes (online transactions without global ordering)

Storage provides *availability of bytes*. Execution correctness remains relationship-local.

**Theorem 7** (Storage does not reintroduce global ordering). *Publishing/fetching objects by content address does not impose a global transaction order. Relationship acceptance remains governed by Acc over relationship chains and local roots.*

*Proof.* Storage acceptance is content-addressed ( $H(o) = \text{Addr}(o)$ ) and availability is quorum-based over a bounded replica set. Execution acceptance  $\text{Acc}$  depends only on adjacency, authorization, and membership proofs into local roots. Neither mechanism requires a universal global order object.  $\square$

## 10 Part IV: Resource model (real networks without time)

We model constrained links using bytes per delivery step, not seconds.

**Definition 23** (Per-edge delivery budgets). For each ordered pair  $(x, y)$  and step  $k$ , define a delivery budget  $\text{Budget}_k(x, y) \in \mathbb{N}$  such that

$$\sum_{(x,y,m) \in \text{Deliver}_k} \text{size}(m) \leq \text{Budget}_k(x, y).$$

**Claim 2** (Environment slows edges, not the whole system). *Budgets can be low on specific edges, reducing progress there, but cannot create global coupling because neither Acc nor quorum fetch references a global serialized object.*

## 11 Part V: Load model (why “not all-in-all” stays balanced)

This section is optional for protocol correctness, but answers the engineering question: *do nodes get overloaded?*

**Assumption 9** (Placement behaves like a PRF). Model  $\text{perm}(\text{Reg}_k, a)$  as a pseudorandom permutation indexed by  $a$ , so that placements are uniformly distributed across nodes. This is a modeling assumption; the protocol itself remains deterministic.

**Theorem 8** (Expected per-node storage load). *Fix a step  $k$  with registry size  $N = |\text{Reg}_k|$  and replication factor  $r$ . Let  $M$  objects be published with independent addresses modeled as uniform. Then the expected number of placements onto any node is  $rM/N$ , and load concentrates around this value with high probability (Chernoff-type concentration).*

*Proof.* Each object chooses  $r$  distinct nodes uniformly from  $N$  (under the PRF model), so indicator variables for “object placed on node  $n$ ” have expectation  $r/N$ . Summing over  $M$  objects gives expectation  $rM/N$ . Standard concentration bounds apply under limited dependence assumptions from sampling without replacement.  $\square$

## 12 Threat model and assumptions

This paper assumes:

- collision resistance of  $H$ ,
- canonical encoding (unique bytes per committed object),
- endpoint authorization unforgeability (abstracted),
- eventual delivery for liveness/repair (no bounds; safety holds without it),
- content integrity by hashing, and redundancy by quorum.

It does *not* assume:

- synchrony or bounded latency,
- trusted servers,
- global consensus or global blocks,
- all-in-all storage,
- time-based validity rules.

## 13 Conclusion

DSM scales because global coupling is removed from both execution and storage. Execution is relationship-local: unrelated activity cannot invalidate your witnesses, and disjoint relationships advance in parallel. Storage is replica-local: objects are placed onto bounded replica sets, verified by quorum fetch, repaired deterministically under churn, and expanded via deterministic capacity signals without turning into a global bottleneck.

### One-line thesis

*DSM scales because it deletes the global object.*

## A Appendix A: Minimal invariants checklist

1. **Canonical bytes:** every committed object has exactly one encoding.
2. **Content addressing:** fetch acceptance requires  $H(o) = \text{Addr}(o)$ .
3. **Replica boundedness:** replica set size  $r$  is fixed and small; no all-in-all.
4. **Quorum:** availability is witnessed by  $q$  identical copies.
5. **Capacity:** admit prevents overfill; ByteCommit makes utilization auditable.
6. **Repair:** replication restored to placement set under eventual delivery.
7. **No clocks:** no rule depends on wall-clock; only step ordering and deterministic predicates.