DJTE: Deterministic Join-Triggered Emissions for the Deterministic State Machine (DSM)

A Proof-Carrying, Consensus-Free Emission System with Shard-Equivalent Global Uniform Sampling

with Deterministic Credit Bundles, Contact-Gated Inboxes, and Precommitment-Driven DoS Minimality

Brandon "Cryptskii" Ramsay

Dec 14 2025

Abstract

DJTE is a deterministic, join-triggered token distribution mechanism designed for DSM: a system that forbids consensus, forbids wall-clock dependence, and treats storage nodes as dumb mirrors. Emission eligibility is unlocked when a device proves it has met the DSM spend-gate (e.g., paid the required set of storage replicas for genesis storage), producing a Join Activation Proof (JAP). DJTE deterministically converts a sequence of activation events into emission events, selecting emission recipients uniformly at random over the global set of activated identities while operating on sharded data structures and without requiring any global roster enumeration. This paper specifies DJTE as a proof-carrying protocol: eligibility, sampling, non-reuse, supply caps, and fork convergence are all enforced by verifiable commitments (Sparse Merkle Trees and append-only accumulators). We formalize and prove (i) determinism, (ii) single-use activation consumption, (iii) shard sampling equivalence to full global uniform sampling, (iv) strict supply upper bounds, and (v) deterministic fork-choice convergence once the same transition set is observed. We also prove that shard selection requires $\Omega(b)$ bits of information to choose among 2^b shards, making the protocol's $O(b)$ shard-descent proof cost information-theoretically minimal; attempted replacements using random-walk selection introduce bias and destroy proof-carrying equivalence guarantees.

Finally, we extend DJTE with (a) a deterministic prepaid *transaction credit bundle* mechanism ("sender-pays credits") that provides economic rate limiting without time, and (b) DSM-native *contact-gated inbox processing* and *precommitment constraints* that dramatically reduce the attack surface for meaningful spam in the first place. We contrast this with blockchain-style systems where global mempools, consensus, and time/fee markets create broad spam externalities and griefing vectors. The resulting system is proof-carrying, deterministic, and structurally hostile to meaningless flooding: in DSM, most arbitrary data cannot even be represented as a valid state-advancing object, and victims can remain online while refusing to process hostile relationships.

Contents

1 System Model and Goals

1.1 DSM constraints (hard requirements)

DJTE is built for DSM-style operation:

- **No consensus and no leader.** Devices verify. Storage nodes store/relay/reject malformed objects. No signatures are required from storage nodes.
- **No wall-clock time.** No timestamps. All state is advanced by deterministic transitions and indices.
- **Proof-carrying objects.** Any verifier can check validity offline from commitments and proofs.
- **Strict-fail verification.** Any missing proof or mismatch is invalid; no "best effort" acceptance.

1.2 Participants, responsibilities, and what users do *not* do

DJTE is intentionally designed so that **end users do not participate in network coordination**. The only "actors" are:

- **End-user devices (DSM clients).** Devices create/hold identities, produce and verify proof-carrying objects, and maintain local deterministic state. Users approve actions in the wallet UI but do not manage shards, select winners, run ordering protocols, or perform manual cryptography.
- **Storage nodes (dumb mirrors).** Storage nodes store and relay objects (by content address) and may reject malformed protobuf objects. They do not vote, order, sign, or decide validity beyond basic structural checks. They are not trusted for correctness.
- **Policy anchors (CPTA/DLV references).** These are content-addressed immutable policy objects referenced by devices during verification. They are not authorities; they are immutable inputs.

Thus, from a user perspective, DJTE "just happens" as part of the wallet's normal operation: unlock spend-gate, receive receipts in the b0x (inbox) when online, and see balances/history updated by verified state transitions. Users do *not* run committees, do *not* watch global mempools, and do *not* perform any manual shard work.

1.3 Problem

We want a deterministic system that:

1. Emits tokens only after a device unlocks the spend-gate, producing a Join Activation Proof (JAP).
2. Selects emission recipients *uniformly over all activated identities* (global uniformity), yet stores data in shards (no global roster scanning).
3. Prevents double-use of Join Activation Proofs (one proof consumes at most one emission transition).
4. Enforces a fixed total supply cap (e.g., $80 \cdot 10^9$ units) with a halving-style schedule.

5. Converges deterministically across forks (devices may observe different transition orders, but converge once they see the same set).

6. Provides robust resistance to meaningless spam by combining precommitment validation, contact gating, strict-fail verification, and optional economic credits — all without time, consensus, or trusted throttles.

1.4 Threat model and security objectives

DJTE assumes:

- Attackers can run arbitrary storage nodes and can withhold/relay objects selectively.
- Attackers can generate many identities if the spend-gate cost is low (Sybil attempts).
- Attackers can attempt to bias recipient selection by manipulating inputs (e.g., crafting identities to target shards).
- Attackers can attempt resource exhaustion by flooding storage nodes or devices with malformed data.
- Attackers may attempt *valid* high-volume traffic if they can afford it.

DJTE must guarantee:

- **Determinism:** Given the same committed state and the same consumed activation, all verifiers compute the same winner.
- **Proof-carrying validity:** All acceptance is local and offline-verifiable from proofs.
- **Global uniformity over eligible identities:** Each activated identity has equal chance per emission event.
- **Non-reuse of activations:** A JAP cannot be consumed twice in valid emissions.
- **Supply cap:** Distributed amount never exceeds the source DLV allocation.
- **Attack-surface minimality:** meaningfully "spamming the network" is structurally difficult because most arbitrary objects fail precommitment and contact-gating constraints *before* they reach costly verification stages.

Sybil resistance is delegated to DSM's spend-gate: DJTE remains mechanically correct even under Sybil behavior, but the *social* meaning of fairness depends on the spend-gate preventing cheap identity grinding.

1.5 Bigger picture positioning

DJTE is an instance of a broader distributed-systems trend: **proof-carrying data structures** replacing "ask the network" semantics. DSM pushes this to an extreme: correctness lives in objects and their proofs, not in agreement processes. The constraints (no time, no consensus, dumb mirrors) are strict, but the payoff is equally strict: offline verifiability, deterministic convergence under shared view, and sharply reduced global spam externalities.

2 Preliminaries

2.1 Byte strings and hash

All identifiers are byte strings. Let $H(\cdot)$ be a cryptographic hash (DSM standard: BLAKE3). Concatenation is denoted $\|$. Domain separation labels are fixed byte prefixes (e.g., `"DJTE.SEED"`).

2.2 Sparse Merkle Tree (SMT)

An SMT maps fixed-length keys to values and supports:

- Membership proof: key $\mapsto$ value present.
- Non-membership proof: key absent (or mapped to default) under a committed root.

2.3 Append-only accumulator

We require an append-only structure supporting:

- **Commitment root** to an ordered sequence of leaves $L[0..n-1]$.
- **Inclusion proof** that leaf $L[i]$ is at index i under the root.

An MMR (Merkle Mountain Range) or append-only Merkle tree qualifies. We abstract it as ACC.

2.4 Deterministic exact-uniform sampling over a range

Given a 256-bit value R and range size $N > 0$, define rejection sampling:

$$\mathsf{UniformIndex}(R, N): \quad \text{let limit} = \left\lfloor \frac{2^{256}}{N} \right\rfloor \cdot N.$$

If $R \geq \text{limit}$, set $R \leftarrow H(\texttt{"DJTE.RESEED"} \| R)$ and retry. Output $R \bmod N$. This yields an exactly uniform integer in $\{0, \ldots, N-1\}$.

2.5 Deterministic "no-op" invalidity

DSM treats *meaningless* operations as structurally invalid. We will use:

Definition 1 (State-advancing transition)**.** *A transition is* state-advancing *if it (i) is authored by the sender identity, and (ii) strictly advances the sender's straight hash chain tip (no duplicates, no replays, no zero-effect updates).*

This is the formal basis for "you can't send zero" and "random bytes do not count as a transaction."

3 Core Objects

3.1 Identity and shard assignment

Definition 2 (IdentityID)**.** *An $\mathsf{IdentityID}$ is a fixed-length byte string representing a DSM identity (e.g., hash of genesis anchor).*

Definition 3 (Shard function)**.** *Fix a shard depth $b \geq 1$. Define:*

$$\mathsf{Shard}(\text{id}) = \text{prefix}_b\big(H(\texttt{"DJTE.SHARD"} \| \text{id})\big) \in \{0,1\}^b.$$

This partitions identities into 2^b shards.

3.2 Join Activation Proof (JAP)

Definition 4 (Join Activation Proof)**.** *A JAP is a proof-carrying object produced by a device when it unlocks the spend-gate. It commits to:*

- id*: the IdentityID being activated,*
- gate*: evidence that required replica payments were satisfied,*
- nonce*: a deterministic per-activation discriminator (e.g., derived from the identity's straight chain tip),*

and yields a unique digest:

$$\text{jap\_hash} = H(\texttt{"DJTE.JAP"} \| \text{id} \| \text{gate} \| \text{nonce}).$$

3.3 Source DLV State (the emission source vault)

All tokens already exist under a CPTA policy and are revealed/distributed deterministically from a locked source DLV.

Definition 5 (Source DLV State)**.** *The source DLV state at emission index e is:*

$$\mathcal{D}_e = \Big(\text{dlv\_tip},\ \text{spent\_root}_e,\ \text{count\_root}_e,\ \{\text{acc\_root}_{e,s}\}_{s\in\{0,1\}^b},\ \text{remaining}_e,\ e\Big)$$

where:

- spent_root_e *is the root of* SpentProofSMT *(consumed JAPs),*
- count_root_e *is the root of* ShardCountSMT *(counts per shard and aggregated subtree sums),*
- $\text{acc\_root}_{e,s}$ *is the commitment root of* Shard Activation Accumulator *for shard s,*
- remaining_e *is remaining undistributed supply under this DLV.*

3.4 Shard Activation Accumulator (SAA)

Definition 6 (SAA)**.** *For each shard s, maintain an append-only accumulator* ACC *whose leaves are activated identities:*

$$\text{SAA}_s : L_s[i] = H(\texttt{"DJTE.ACTIVE"} \| \text{id}_i).$$

The accumulator provides $\text{acc\_root}_{e,s}$ *and inclusion proofs of $(i, L_s[i])$.*

3.5 ShardCountSMT: proof-carrying global counts

Definition 7 (ShardCountSMT)**.** *Define a complete binary tree over shard prefixes up to depth b. Each node corresponds to a prefix p of length $k \le b$ and stores:*

$$\text{count}(p) = \sum_{s \text{ extends } p} |L_s|.$$

Leaves are $p = s$ of length b and store $\text{count}(s) = |L_s|$*. All nodes are committed in an SMT keyed by p (as bytes), yielding root* count_root_e*.*

This permits:

- proving total activated identities $N = \text{count}(\epsilon)$ (empty prefix),
- deterministically mapping a global rank $k \in [0, N)$ to a shard and local index using $O(b)$ node proofs.

3.6 SpentProofSMT

Definition 8 (SpentProofSMT)**.** *An SMT mapping* jap_hash $\mapsto 1$ *if consumed, else absent. Root at state* e *is* spent_root_e*.*

3.7 Emission receipt

Definition 9 (Emission Receipt)**.** *An emission event at index* e *produces a receipt:*

$$\text{rcpt}_e = H(\texttt{"DJTE.RCPT"} \| e \| \text{winner\_id} \| \text{amount}_e \| \text{jap\_hash})$$

and includes:

- *proofs used for winner selection,*
- *proofs that* jap_hash *was unspent before and marked spent after,*
- *supply update evidence,*
- *(optional) compact witnesses (multi-proofs) enabling efficient batch verification.*

4 Policy and Emission Schedule

4.1 Fixed supply cap and halving epochs

Let S_{total} be total supply allocated to this DLV (e.g., $80 \cdot 10^9$ units). DJTE emits from *remaining* supply; it never creates new units.

We define a halving schedule as a policy tuple:

$$\Pi = (S_{\text{total}}, b, E, M_0, r_0)$$

where:

- b is shard depth,
- $E = 16$ is number of epochs,
- M_0 is max emissions in epoch 0,
- r_0 is per-emission amount in epoch 0,
- epoch i has $(M_i, r_i) = (2^i M_0, \lfloor r_0/2^i \rfloor)$.

To enforce a strict cap independent of join count, the protocol enforces:

$$\text{amount}_e = \min\left(r_{\mathsf{epoch}(e)}, \text{remaining}_e\right),$$

and if $\text{remaining}_e = 0$ then $\text{amount}_e = 0$ and the DLV emits nothing thereafter.

Lemma 1 (Supply upper bound)**.** *For all reachable states, total distributed supply is* $\leq S_{total}$*.*

Proof. remaining starts at S_{total} and is updated by $\text{remaining}_{e+1} = \text{remaining}_e - \text{amount}_e$ with $\text{amount}_e \leq \text{remaining}_e$. Thus remaining never goes negative and distributed amount never exceeds S_{total}. □

5 Protocol Algorithms (Deterministic)

5.1 Activation: incorporating a Join Activation Proof

When a verifier accepts a JAP as valid (per DSM spend-gate rules), it updates three structures deterministically:

1. **SAA append:** let $s = \mathsf{Shard}(\text{id})$ and append leaf $L_s[n_s] = H(\texttt{"DJTE.ACTIVE"} \| \text{id})$.
2. **ShardCountSMT update:** increment $\text{count}(s)$ and all ancestors up to root ϵ.
3. **(Optional) audit index:** store the JAP object by content address for retrieval; this is not a consensus primitive.

All updates are committed in the next DLV state by changing $\text{acc\_root}_{e,s}$ and count_root_e accordingly.

5.2 Winner selection at emission index e

Winner selection is deterministic and proof-carrying.

Inputs. Current source DLV state $\mathcal{D}_e$ and a consumed JAP with hash jap_hash.

Step 1: derive seed.

$$R_0 = H(\texttt{"DJTE.SEED"} \| \text{dlv\_tip} \| e \| \text{jap\_hash}).$$

Step 2: obtain total eligible count. Verifier requires a proof from $\mathsf{ShardCountSMT}$ that:

$$N = \text{count}(\epsilon).$$

If $N = 0$, winner selection is undefined and the emission transition *must* set $\text{amount}_e = 0$.

Step 3: sample global rank uniformly. Compute:

$$k = \mathsf{UniformIndex}(R_0, N) \in \{0, \ldots, N-1\}.$$

Step 4: map global rank to shard and local index. Using count proofs along the prefix tree:

- Start at prefix $p = \epsilon$ and rank k.
- For each depth $d = 0..b-1$, query children $p0$ and $p1$ counts with SMT proofs.
- If $k < \text{count}(p0)$, set $p \leftarrow p0$; else set $k \leftarrow k - \text{count}(p0)$ and $p \leftarrow p1$.
- After b steps, p is shard s and remaining k is local index i within shard s.

Step 5: prove winner identity at (s, i). Verifier requires an inclusion proof from SAA for shard s that leaf at index i is:

$$L_s[i] = H(\texttt{"DJTE.ACTIVE"} \| \text{winner\_id}).$$

Winner is winner_id.

5.3 Consuming a JAP exactly once (SpentProofSMT)

To advance from $\mathcal{D}_e$ to $\mathcal{D}_{e+1}$ with a nonzero emission, the transition must carry:

- **Non-membership proof** under spent_root_e that jap_hash is absent.
- **Membership proof** under spent_root_{e+1} that $\text{jap\_hash} \mapsto 1$ is present.

5.4 State transition validity

A candidate next state $\mathcal{D}_{e+1}$ is valid iff all hold:

1. **Index increment:** $e_{next} = e + 1$.
2. **Spent update correctness:** spent_root_{e+1} is spent_root_e plus insertion of jap_hash, and proofs verify.
3. **Winner selection correctness:** the rank derivation and shard mapping proofs verify under count_root_e, and SAA inclusion verifies under $\text{acc\_root}_{e,s}$.
4. **Supply update:** $\text{remaining}_{e+1} = \text{remaining}_e - \text{amount}_e$ with $\text{amount}_e \leq \text{remaining}_e$.
5. **Receipt binding:** rcpt_e matches the provided values and proofs.

6 Bootstrap Regime: Very Few Activated Identities

6.1 Correctness when N is small

At system start, the global eligible population is small:

- If $N = 0$, emission must be a no-op ($\text{amount}_e = 0$). Any nonzero winner claim is invalid because it cannot supply valid rank mapping and inclusion proofs.
- If $N > 0$ but many shards are empty, the ShardCountSMT descent routes ranks only into nonempty subtrees; empty shards get exactly zero probability mass, which is correct for global-uniform sampling over individuals.

6.2 History, balances, and user experience

Because receipts are proof-carrying, a DSM client can update history/balances deterministically after local verification. There is no "ask the network" step. Early bootstrap does not require special coordination; it only requires strict-fail handling of $N = 0$.

7 Fairness Under Shard Imbalance

7.1 What "fair" means in DJTE

DJTE fairness target is **global uniformity over activated identities**. If shard s contains $|L_s|$ identities and total $N = \sum_s |L_s|$:

$$\Pr[\text{winner in shard } s] = \frac{|L_s|}{N}.$$

This is exactly "each identity has probability $1/N$."

7.2 Imbalance is expected and not unfair

Shard skew is not unfair; it is correct under global-uniform semantics. "Equalizing shards" would violate uniformity over individuals.

7.3 The real fairness risk: shard targeting and identity grinding

Risk comes from malleable identity creation. Mitigation is DSM spend-gate and device/identity binding, making grinding expensive. DJTE itself remains correct; the spend-gate provides the anti-grind economics.

8 Rigorous Guarantees

8.1 Determinism

Theorem 1 (Determinism)**.** *Given the same starting state $\mathcal{D}_e$ and the same consumed* jap_hash*, any verifier computes the same winner and the same next state $\mathcal{D}_{e+1}$.*

Proof. All steps are fixed functions of committed roots and domain-separated hashes: R_0, exact-uniform sampling, deterministic rank-to-shard descent, deterministic inclusion proofs, deterministic SMT updates. No timestamps, no nondeterministic branching. □

8.2 Single-use of Join Activation Proofs

Theorem 2 (No double-consumption)**.** *No JAP hash can be consumed in two distinct valid emission transitions descending from the same prior state.*

Proof. A valid transition requires non-membership of jap_hash under spent_root_e. After consumption, spent_root_{e+1} includes it permanently. Any later attempt fails the required non-membership proof. □

8.3 Global uniformity from sharded storage

Definition 10 (Global activated multiset)**.** *Let the global activated list be the concatenation of shard lists in lexicographic shard order:*

$$G = L_{s_0} \| L_{s_1} \| \cdots \| L_{s_{2^b-1}}, \quad N = |G| = \sum_s |L_s|.$$

Lemma 2 (Rank mapping correctness)**.** *Given correct ShardCountSMT counts, the descent maps each $k \in [0, N)$ to exactly one (s, i) such that $G[k] = L_s[i]$.*

Proof. At each prefix p, left subtree size is $\text{count}(p0)$ and right subtree size is $\text{count}(p1)$. The descent preserves rank by subtracting $\text{count}(p0)$ when crossing right. After b steps, the selected shard and residual rank uniquely identify $G[k]$. □

Theorem 3 (Global uniformity)**.** *If k is uniform in $\{0, \ldots, N-1\}$, then DJTE selects a uniformly random element of G.*

Proof. By Lemma, the algorithm returns $G[k]$. Uniform k implies uniform selection over positions of G. □

9 Proof Size, Verification Cost, and Why It Does Not Blow Up

9.1 Per-emission proof components

A receipt contains:

- Spent non-membership under spent_root_e and membership under spent_root_{e+1}.
- Count descent openings (conceptually $O(b)$).
- SAA inclusion proof at index (s, i).

Proof size is independent of N, scaling as:

$$O(\log(\text{SMT depth})) + O(b) + O(\log |L_s|).$$

9.2 Multi-proofs and witness compression

Production implementations must pack count openings as multi-proofs, eliminating repeated siblings/ancestors. Formally, a multiproof for a set of queried nodes under one root can be represented as:

- the set of revealed node values,
- the minimal set of sibling hashes required to recompute the root.

This yields one compact witness for the entire descent path rather than $O(b)$ independent witnesses.

9.3 Batch verification and caching

Devices can batch-verify receipts and cache ShardCountSMT node values for prefixes, since they only change when new activations occur under those subtrees. This makes "frequent emissions" operationally cheap: verification is dominated by a small number of hash recomputations and a bounded set of proof checks.

9.4 Receipt growth vs bounded single-receipt cost

History may grow large, but *one* receipt remains bounded/logarithmic to verify. DSM can additionally support policy-controlled pruning where raw receipts are discarded after their effects are incorporated, while retaining committed roots (audit preferences determine how much to keep).

10 Why Random Walks Do Not Replace Shard Descent

10.1 The $O(b)$ descent cost is information-theoretically minimal

Theorem 4 (Shard choice lower bound)**.** *Any procedure that selects one shard among 2^b shards requires $\Omega(b)$ bits of decision information.*

Proof. Distinguishing 2^b outcomes requires at least b bits of information. □

10.2 Random-walk selection breaks proof-carrying equivalence

Random walks require a step policy and mixing guarantees; exact global-uniform equivalence is not proof-carrying without additional heavy machinery. In particular, "random walk until mixed" introduces:

- an implicit time/iteration parameter (a policy knob),
- a stationary-distribution proof burden,
- bias if mixing is insufficient or adversarially structured.

DJTE therefore rejects random-walk selection as a replacement for rank descent.

11 Contact-Gated Inboxes and Precommitment: Why Meaningful Spam is Hard

11.1 Precommitment constraints

DSM transitions are not arbitrary messages. In DSM, a valid incoming object is constrained by precommitted structure:

- A receiver's relationship-specific straight hash chain evolves only by valid transitions that match expected preimage structure and known commitment context.
- Random bytes cannot be "appended" to a precommitted chain: verification fails immediately under strict-fail rules.
- Many message classes require binding to prior commitments (known chain tips, expected indices, prior receipts, or policy anchors), defeating generic garbage injection.

Definition 11 (Relevance predicate)**.** *Let* $\mathsf{Rel}(\mathrm{obj}, \mathrm{ctx}) \in \{0, 1\}$ *be a deterministic predicate that checks whether an object references the receiver's expected commitment context (e.g., known tips/indices/policy anchors for that relationship). Objects with* $\mathsf{Rel} = 0$ *are rejected without deep proof evaluation.*

Claim 1 (Precommitment excludes arbitrary spam early)**.** *For an attacker lacking the receiver's commitment context, the probability that a random object satisfies* $\mathsf{Rel}(\mathrm{obj}, \mathrm{ctx}) = 1$ *is negligible under standard hash binding assumptions.*

11.2 Contact gating (relationship-specific acceptance)

DSM's operational model is relationship-specific:

- Devices accept and process inbox objects only for relationships/contacts they have established.
- A device can be online without syncing or processing any particular contact's inbox stream.
- A device can reject or block a contact and thereby stop ingesting their objects entirely, without impacting other relationships or global correctness.

Definition 12 (Contact gating)**.** *Let* $\mathsf{Allow}(\mathrm{sender}, \mathrm{receiver})$ *be a deterministic predicate derived from the receiver's local contact set. Inbox processing for a sender is performed only if* $\mathsf{Allow} = 1$*.*

Lemma 3 (Victim opt-out kills per-contact flooding)**.** *If a victim sets* $\mathsf{Allow}(\mathrm{attacker}, \mathrm{victim}) = 0$*, then attacker-originated objects are not processed by the victim regardless of the victim being online.*

Proof. By definition of contact gating, inbox processing does not run for disallowed contacts; therefore no per-contact processing cost is incurred. □

11.3 No global mempool externality

Unlike blockchain systems with global mempools, DSM does not expose a single shared global queue where anyone can force everyone to compete for inclusion. DJTE receipts are proof-carrying; devices do not need global ordering, and storage nodes do not impose a global fee market. This structurally removes the classic "spam the mempool to grief everyone" channel.

11.4 "You can't send zero" and meaninglessness filtering

DSM can enforce that meaningless operations are not valid transitions:

- If an operation has zero-value effect or does not advance required commitments, it is invalid and rejected.
- Because debit rules and acceptance are tied to state-advancing transitions, "spam" that does nothing cannot be represented as an accepted object and therefore cannot impose systemic cost.

11.5 Offline bilateral transfers

Offline bilateral transfers (e.g., Bluetooth) are inherently resistant to network-style spam:

- physical proximity is required,
- both parties must consent and validate,
- there is no global broadcast surface to flood.

Therefore, the high-risk spam surface is primarily *remote valid traffic*, not local bilateral exchange.

12 Deterministic Credit Bundles: Economic Rate Limiting Without Time

12.1 Motivation

Even though DSM's structure already minimizes meaningful spam vectors, credit bundles add a deterministic economic backstop against *valid* high-volume traffic aimed at storage/relay capacity. Importantly, this is sender-only: victims cannot have credits drained by receiving traffic.

12.2 Credit bundle policy (protocol-level constant)

Definition 13 (Credit Bundle Policy)**.** *A CPTA-anchored policy object* Π_{cred} *defines:*

$$\Pi_{\text{cred}} = (C_{\text{bundle}},\ \mathit{DebitRule},\ \mathit{RefillRule})$$

where C_{bundle} *is a* globally verifiable protocol-level constant *(baseline* $C_{\text{bundle}} = 1000$*), with no time windows and no storage-node discretion.*

12.3 Credit counter in identity state

Definition 14 (Credit Counter)**.** *Each identity maintains* credits(id) $\in \mathbb{N}$ *inside its deterministic committed state.*

12.4 Debit rule (sender-pays)

Definition 15 (DebitRule: accepted state-advancing transitions)**.** *A transition authored by* id *debits exactly one credit iff:*

- *it is sender-authored,*
- *it verifies and is accepted,*
- *it advances the sender's straight hash chain tip (Definition 1).*

Inbound receipts/emissions, malformed objects, duplicates, replays, and non-advancing objects debit zero.

Lemma 4 (No victim credit drain)**.** *An adversary cannot reduce victim credits via inbox flooding.*

Proof. Credits debit only on sender-authored accepted state-advancing transitions. Inbox objects do not debit. Therefore victim credits cannot be drained remotely. □

12.5 Refill rule (spend-gate top-up)

Definition 16 (RefillRule)**.** *A refill is a deterministic sender-authored transition that includes proof of satisfying spend-gate/refill conditions and sets* credits(id) $:= C_{\text{bundle}}$*. If* credits $= 0$*, any further sender-authored state-advancing transition is invalid unless it is a refill.*

12.6 Why $C_{\text{bundle}} = 1000$ is the correct baseline

DSM needs a *non-annoying* bundle that still makes sustained valid flooding expensive, without time-based throttles. 1000 is the correct baseline because:

- It is high enough that typical humans essentially never notice it (normal usage rarely hits 1000 state-advancing sends between refills).
- It is low enough that *valid* sustained flooding forces frequent refills, converting throughput abuse into a direct economic burn.
- Heavy integrators (exchanges/apps) can deterministically automate refill transitions; this is operationally simple because there is no monthly schedule and no time window semantics.
- It is enforceable at protocol level: storage nodes cannot "grant someone a million" because credits are checked by verifiers, not by storage nodes.

Thus, the protocol-level answer is: **start at** 1000 **credits per bundle**. Tune later only via policy anchors, but always as globally verifiable constants.

12.7 What would a spammer gain?

In DSM, spamming does *not* buy consensus influence (there is none), does *not* buy global ordering advantage, and does *not* force victims to process data (contact gating). The only plausible "gain" is localized resource pressure on storage/relay capacity. Credit bundles (plus cheap gating) directly price that behavior, while precommitment/contact constraints already limit its reach.

13 CPU/Parsing DoS: Deterministic Staged Verification and Bounded Work

13.1 Staged verification (cheap-first, strict-fail)

Clients and storage nodes apply deterministic staged verification:

1. Structural gate: protobuf decode, envelope version, size bounds, required fields.
2. Content-address gate: declared hash matches bytes.
3. Relevance gate: $\mathsf{Rel}(\text{obj}, \text{ctx}) = 1$ (precommitment-bound).
4. Contact gate: $\mathsf{Allow}(\text{sender}, \text{receiver}) = 1$.
5. Full proof verification only after gates pass.

This ensures invalid floods are rejected cheaply and do not create systemic externalities.

13.2 Deterministic bounded inbox processing (no time)

Definition 17 (Inbox processing bound)**.** *A device processes at most K candidate objects per inbox-check action; remaining objects are deferred. K is a deterministic local constant (UI/engine policy), not a time-based throttle.*

Lemma 5 (Bounded worst-case per-check work)**.** *Under bounded inbox processing, a single inbox-check action performs $O(K)$ parsing and at most $O(K)$ deep verifications, regardless of how many objects exist remotely.*

Proof. Immediate from the definition: the device selects at most K candidates, applies cheap gates, and only then performs deep verification for survivors. Objects beyond K are not touched in that action. □

14 Fork behavior and deterministic convergence

Definition 18 (Valid DLV state DAG)**.** *States are nodes; an edge $\mathcal{D}_e \to \mathcal{D}_{e+1}$ exists if the transition verifies. Forks can exist.*

Definition 19 (Canonical tip rule)**.** *Choose the tip maximizing e; break ties by lexicographically smallest* dlv_tip*.*

Theorem 5 (Eventual convergence under shared view)**.** *If two verifiers have the same set of valid states, they compute the same canonical tip.*

Proof. The rule is a pure function of the set. □

15 Contrast With Blockchain-Style Systems

15.1 Global queues vs relationship-specific verification

Blockchains expose global mempools and consensus ordering. Any actor can submit transactions that impose validation/bandwidth costs on many participants, even if ultimately not included. DSM avoids this by:

- eliminating consensus and global ordering,
- making finality local (device-verifiable),
- pushing relevance to relationship-specific precommitments and contact gating.

15.2 Fee markets and time dependence vs deterministic policy

Blockchain congestion control is often a fee market coupled to time/blocks. DSM forbids time dependence and uses deterministic policy anchors (CPTA) plus proof-carrying validation. Credit bundles (if enabled) are deterministic and sender-funded without time windows. There is no block cadence, no timestamp games, and no "pay to get into the next block" semantics.

15.3 Spam incentives and griefing

In blockchain systems, spam can be used for censorship-by-congestion, MEV externalities, and mempool griefing. In DSM:

- there is no global mempool to congest,
- inbox processing is optional per contact,
- irrelevant objects fail precommitment constraints,
- sender-pays credits make sustained valid spam economically expensive,
- victims can remain online while refusing to sync hostile relationships.

15.4 What DSM "buys" by being strict

DSM's strict constraints buy guarantees that are hard to obtain in blockchain-style systems:

- offline-verifiable correctness without global agreement,
- minimal global externalities from arbitrary third parties,
- deterministic convergence once the same set of proofs is observed,
- bounded verification costs per object and per inbox-check action.

16 Implementation Mapping to DSM (Operational Rigor)

16.1 Wire format and determinism discipline

All DJTE objects use DSM envelopes with protobuf-only serialization. No timestamps, no nondeterministic fields, no reliance on transport ordering.

16.2 Verification pipeline (strict-fail)

A DSM device verifying a DJTE transition must:

1. Apply staged gates (structural, hash, relevance, contact).
2. Verify DJTE proofs against spent_root_e, count_root_e, and $\text{acc\_root}_{e,s}$.
3. Recompute R_0, compute uniform k, derive (s, i), verify SAA inclusion.
4. Verify supply decrement and receipt binding.
5. Enforce any protocol-level credit bundle rule for sender-authored state-advancing transitions.
6. Accept iff all checks pass.

16.3 Storage nodes as dumb mirrors

Storage nodes store and relay objects by content address, apply cheap structural gates, and do not decide correctness.

16.4 b0x (inbox) dissemination model

Devices check b0x when online, selectively per contact. They validate locally and update history/balances deterministically. Finality does not require going online.

17 Additional Notes and Practical Parameter Guidance

17.1 Choosing shard depth b

Choose $b \in [16, 20]$ to fix dispersion and keep descent bounded; compress witnesses via multi-proofs. The engineering goal is *not* to eliminate b (impossible without losing information), but to keep b fixed and compress its proof witnesses.

17.2 DoS considerations (full summary)

DJTE/DSM attack surface is minimized because:

- Random data cannot bind to precommitted hash chains and is rejected.
- Contact gating lets users ignore/block relationships without affecting other operations.
- There is no consensus/mempool externality, so "spamming the network" is not globally amplifying.
- Optional credit bundles impose an economic burn on accepted sender-authored state-advancing traffic.
- Staged verification ensures invalid floods are cheaply rejected.
- Offline bilateral transfers are physically consented and naturally rate-limited.

17.3 What DJTE is not

DJTE is not consensus. Forks can exist until devices learn more states; convergence is deterministic once the same set is known. DJTE is not a social oracle; it is a mechanical fairness mechanism under a Sybil-resistant spend-gate.

18 Conclusion

DJTE provides deterministic, proof-carrying, join-triggered emissions for DSM with global-uniform sampling over activated identities while using sharded storage and no global roster enumeration. Shard descent is $O(b)$ and information-theoretically minimal, with random-walk alternatives rejected for bias and non-proof-carrying equivalence. Beyond emissions, DSM's precommitment structure and relationship-specific contact gating make meaningful spam difficult: arbitrary data cannot bind to committed chains, and users can selectively ignore or block contacts without affecting other relationships or global correctness. Optional deterministic credit bundles add a sender-funded economic backstop against valid high-volume traffic without time windows or node discretion. Compared to blockchain systems, DSM avoids global mempool externalities, consensus-driven congestion, and time-based fee markets, achieving strong guarantees via proof-carrying objects and local verification.

A Appendix A: Minimality vs "Linear b" concerns

The shard descent cost being linear in b is not a design flaw; it is the necessary cost of selecting among 2^b possibilities with proof-carrying determinism. You can compress witnesses (multi-proofs), batch verification, and cache node values, but you cannot eliminate the information requirement without changing the semantics (and breaking global uniformity equivalence).

B Appendix B: Summary of what end users do

End users:

- unlock spend-gate once (wallet handles proofs),
- optionally top up credits when they run out (wallet can automate),
- accept/verify receipts locally (automatic),
- block/ignore contacts (UI-level control).

End users do *not* run ordering, consensus, shards, committees, or manual cryptographic procedures.