

Nyxanic: Self Sustaining Stablecoin

using control theory

Maintaining a \$1.00 Peg with Minimal Idle Capital
Through Time-Driven Flow Control

Whitepaper v1.1

<https://github.com/morphym/nyxanic>

Abstract

We present *Nyxanic* — a self-sustaining stablecoin architecture on Solana that maintains a hard \$1.00 peg without relying on external incentives, capital subsidies, or oversized collateral reserves. Unlike conventional stablecoins, which respond to peg deviation by deploying capital, Nyxanic responds by controlling *flow* — adjusting fees, spreads, throttling, routing, supply, and ultimately suspending transactions in a strict severity-driven ladder. The architecture is governed entirely by Control Theory and operates on two principles: **time is the dictating factor**, and **stress generates the energy required to resist it**. The system consists of two on-chain Solana contracts — a Token Contract (the body) and a Math Contract (the brain) — communicating atomically via Cross-Program Invocation. The mathematics is closed-form, deterministic, and domain-independent. A reference implementation has been built and deployed to Solana devnet; deployed addresses, measured compute cost, and observed behavior are reported in Sections 9–12. Source: <https://github.com/morphym/nyxanic>.

Contents

1	Introduction	4
1.1	The Problem	4
1.2	The Vision	4
2	Mathematical Architecture	4
2.1	Foundational Definitions	4
2.2	Error Signal	4
2.3	Severity Function	5
2.4	Direction of Transaction	5
2.5	Severity Velocity	5
2.6	The Response Ladder	5
2.7	Ladder Transition Function	6
2.8	Intensity Dynamics	6
2.9	Reordering at Medium Severity	6
2.10	Dynamic Fee Function	7
2.11	Terminal Condition	7
2.12	Parameter Summary	7
3	System Architecture on Solana	7

3.1	Two-Contract Design	7
3.2	Communication	8
3.3	Account Layout	8
3.4	Concurrency Property	8
4	The Heartbeat Protocol	9
4.1	The <code>evaluate_transaction</code> Instruction	9
4.2	The Directive Response	9
4.3	Defensive Fallback	9
5	Self-Sustainability	9
6	Birth Sequence	10
7	Properties	10
7.1	Capital Efficiency	10
7.2	Cost Frugality	10
7.3	Adversarial Resistance	11
7.4	Trustlessness	11
7.5	Determinism	11
7.6	Observability	11
7.7	Trustless Price Source	11
8	Domain Independence	11
9	Reference Implementation	12
9.1	Repository Layout	12
9.2	Math Library (<code>organism-math</code>)	12
9.3	Body — Token Contract (<code>organism-token</code>)	13
9.4	Brain — Math Contract (<code>organism-brain</code>)	13
9.5	Implementation Notes Beyond the Original Specification	13
10	Measured Compute Cost	14
11	Observations from Devnet	14
11.1	Severity (§2.3)	14
11.2	Hysteresis (§2.7)	15
11.3	Permutation (§2.9)	15
11.4	6 Supply Correction	15
11.5	7 Collateral Ratio Enforcement	15
11.6	Authority Handoff (Step 9)	15
11.7	Asymmetric Recovery (Limitation)	16
12	Test Currency: NTT (Nyxanic Test Token)	16
12.1	Hard Constraints	16
12.2	Mechanism	16
12.3	Errors	16
13	Tooling	17
13.1	Random Multi-Account Test	17
13.2	Interactive Terminal	17

14 Conclusion	17
A Glossary of Symbols	18
B Compute Cost Comparison (Measured)	18
C Deployed Addresses (Solana Devnet)	18

1 Introduction

1.1 The Problem

Stablecoins on Solana — and on every blockchain to date — exhibit three structural lackings:

1. **Lack of self-sustainability.** Most designs rely on external incentives (token emissions, governance subsidies, foundation reserves). When subsidies dry up, peg integrity erodes.
2. **Lack of capital-efficient peg architecture.** Existing systems either over-collateralize heavily (capital inefficient) or under-collateralize and fail under stress (UST/LUNA, IRON, BEAN). No design maintains a hard \$1.00 peg with *minimal idle capital*.
3. **Lack of native unified liquidity.** Solana’s stablecoin liquidity is fragmented across Raydium, Orca, Jupiter, and others, requiring complex routing and creating slippage at scale.

This paper addresses the first two. The third is left to future work and protocol-level integration.

1.2 The Vision

We propose treating the stablecoin not as a passive financial instrument but as a **living organism**: a system that perceives disturbances, reacts proportionally, escalates intelligently, and rests when calm — all without external intervention.

Nyxanic does not spend capital to defend its peg. It controls *flow*. It applies friction where helpful, opens channels where stabilizing, and ultimately freezes itself before failing.

The architecture rests on a single insight from Control Theory:

If you can measure the error, you can correct it.

Combined with the self-funding insight:

The harder the system is attacked, the more energy it generates to defend itself.

2 Mathematical Architecture

2.1 Foundational Definitions

Let $P(t)$ denote the observable price of the stablecoin at time t , and let $P^* = 1.00$ denote the target peg. All control derives from the relationship between $P(t)$ and P^* .

2.2 Error Signal

The error signal is the signed deviation of price from peg:

$$e(t) = P^* - P(t) \tag{1}$$

The sign of $e(t)$ encodes corrective direction: positive means upward force is required, negative means downward, zero means equilibrium.

2.3 Severity Function

The severity function $s(t) \in [0, 1]$ maps error magnitude to a bounded activation signal:

$$s(t) = \frac{|e(t)|^\alpha}{|e(t)|^\alpha + \delta^\alpha} \quad (2)$$

This is a Hill function, drawn from biological dose-response systems. It satisfies:

$$\lim_{|e| \rightarrow 0} s = 0, \quad s|_{|e|=\delta} = \frac{1}{2}, \quad \lim_{|e| \rightarrow \infty} s = 1$$

The parameter δ defines the sensitivity point (error magnitude at half-activation). The parameter α defines the sharpness of transition between dormancy and full activation.

2.4 Direction of Transaction

Let $d \in \{-1, +1\}$ denote the direction of an incoming transaction:

- $d = +1$: transaction is *destabilizing* (pushes P further from P^*)
- $d = -1$: transaction is *stabilizing* (pushes P toward P^*)

2.5 Severity Velocity

Nyxanic observes not only severity but its motion through time:

$$\Delta s(t) = s(t) - s(t - \Delta t) \quad (3)$$

This is the **second sense**. Severity tells it where it is; velocity tells it where it is going. Time enters the architecture here.

2.6 The Response Ladder

Nyxanic possesses an ordered set of $N = 8$ controllable variables, arranged from lightest to heaviest:

$$\mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_8\} \quad (4)$$

Rung	Variable	Cost
ℓ_1	Dynamic Fee Rate	None
ℓ_2	Spread Width Control	None
ℓ_3	Transaction Sequencing	Trivial
ℓ_4	Transaction Speed (throttle)	Trivial
ℓ_5	Liquidity Routing	Reserve movement
ℓ_6	Mint and Burn Rate	Supply change
ℓ_7	Collateral Ratio Adjustment	Reserve binding
ℓ_8	Transaction Stopping	Suspension

The ladder position is denoted $r(t) \in \{0, 1, \dots, 8\}$, with $r = 0$ meaning the system is at rest. Each rung has a severity threshold θ_r with:

$$0 < \theta_1 < \theta_2 < \dots < \theta_8 \leq 1 \quad (5)$$

2.7 Ladder Transition Function

The ladder evolves under transition function $\mathcal{T}$:

$$r(t + \Delta t) = \mathcal{T}(r(t), s(t), \Delta s(t), \tau(t)) \quad (6)$$

with three rules:

Climb: $r(t + \Delta t) = r(t) + 1$ if $\Delta s \geq 0 \wedge \tau \geq \tau_{\text{wait}} \wedge s \geq \theta_{r(t)}$

Hold: $r(t + \Delta t) = r(t)$ if $|\Delta s| \approx 0 \wedge s \in [\theta_{r-1}, \theta_r]$

Descend: $r(t + \Delta t) = r(t) - 1$ if $\Delta s \leq 0 \wedge s < \theta_r \wedge \tau \geq \tau_{\text{cool}}$

The ladder is climbed and descended one rung at a time. No jumps are permitted. The asymmetry $\tau_{\text{cool}} > \tau_{\text{wait}}$ creates hysteresis, preventing oscillation between rungs. The descent rule uses $\Delta s \leq 0$ rather than the strict $<$ from earlier drafts: when severity hits zero and stays there, $\Delta s = 0$, and a strict inequality would freeze the ladder forever (see Section 11).

2.8 Intensity Dynamics

Each active rung carries an intensity $I_r(t) \in [0, 1]$. The complete intensity dynamics:

$$\boxed{I_r(t + \Delta t) = I_r(t) + \eta(1 - I_r(t)) \mathbf{1}_{[\Delta s_{\text{obs}} \geq 0, s \geq \theta_{\text{high}}]} - \gamma I_r(t) \mathbf{1}_{[\Delta s_{\text{obs}} < 0, \tau \geq \tau_{\text{cool}}]}} \quad (7)$$

where $\Delta s_{\text{obs}} = s(t_0 + \tau_{\text{obs}}) - s(t_0)$ is severity change measured across the observation window. Two opposing forces: growth when failing, decay when succeeding. Time gates both directions.

2.9 Reordering at Medium Severity

At medium severity ($\theta_{\text{low}} \leq s < \theta_{\text{high}}$) with $\Delta s_{\text{obs}} \geq 0$, Nyxanic does not increase intensity. Instead it permutes the active set:

$$\mathcal{A}(t + \Delta t) = \Pi(\mathcal{A}(t)) \quad (8)$$

with Π drawn deterministically from a seed mixing slot, severity, and the permutation count. Once $r!$ permutations have been tried (where r is the active rung count), the ladder must escalate via the climb rule.

The reference implementation uses a Fisher–Yates shuffle on the active rung array, seeded by:

$$\text{seed} = \text{slot} \oplus \text{ROL}_{13}(s) \oplus (\text{count} \cdot \varphi_{64})$$

where $\varphi_{64} = \text{0x9E3779B97F4A7C15}$ is the 64-bit fractional golden ratio. The shuffle uses a linear congruential PRNG seeded by this value. Determinism is required for Solana validator agreement.

Randomness is essential: it breaks adversarial prediction. An attacker cannot anticipate which organ will dominate next.

2.10 Dynamic Fee Function

The lightest organ (rung ℓ_1) is the dynamic fee adjustment:

$$f(t) = f_0 + s(t) \cdot \lambda \cdot d \quad (9)$$

This costs zero capital. It shapes behavior through price signals alone. Destabilizing transactions are taxed; stabilizing transactions are rewarded.

2.11 Terminal Condition

The final rung activates only when all lighter rungs have been exhausted, all reorderings tried, and intensity has reached unity, with severity still failing to fall:

$$r = 8 \wedge I_8 \rightarrow 1 \wedge \Delta s_{\text{obs}} \geq 0 \Rightarrow \text{flow suspended} \quad (10)$$

Transaction halt is the cheapest action computationally and the most preserving for the peg. Suspension is lifted when $\Delta s_{\text{obs}} < 0$ persists across τ_{cool} .

2.12 Parameter Summary

Parameter	Name	Role
δ	Sensitivity point	Error at half-activation
α	Sharpness	Steepness of severity transition
λ	Fee sensitivity	Aggressiveness of fee adjustment
f_0	Base fee	Flat cost at equilibrium
τ_{wait}	Patience	Delay before climbing
τ_{cool}	Cooldown	Delay before descending
τ_{obs}	Observation window	Time to judge action effect
η	Escalation rate	Speed of intensity growth
γ	Descent rate	Speed of intensity decay
I_{min}	Minimum intensity	Activation floor per rung
θ_r	Rung thresholds	Severity boundaries

3 System Architecture on Solana

3.1 Two-Contract Design

Nyxanic is implemented as two on-chain Solana contracts:

- **Contract A — Token Contract (the body).** Holds the existence of the stablecoin. Manages all flow of value: mint, burn, transfer, fee application, flow controls. Interacts with users, pools, and other Solana programs.
- **Contract B — Math Contract (the brain).** Holds perception and decision-making. Computes severity, ladder transitions, intensity dynamics, permutations. Touches no value.

The split principle:

Contract A holds what is. Contract B holds what to do about it.

3.2 Communication

Communication is exclusively via Solana Cross-Program Invocation (CPI), unidirectional from A to B. Contract B is reactive: it has no autonomous triggers. Contract A invokes B during every user transaction, atomically, within a single Solana transaction.

The off-chain computation alternative (encrypted memo channel) was rejected on the grounds that:

- It introduces multi-block latency violating the architecture’s timing assumptions
- It requires liveness guarantees on an off-chain machine
- It demands cryptographic key management
- Compute cost on-chain is in fact negligible (under 2% of Solana’s budget; see Section 10)

3.3 Account Layout

Each contract owns a small set of Program Derived Address (PDA) accounts:

Contract A (Token Contract):

A1: Mint Account	(SPL Mint, supply + decimals)
A2: Reserve Vault	(SPL Token Account, collateral)
A3: Body State Account	(current directive, prices, throttle window)
A4: Fee Collector	(SPL Token Account, retained fees)
A5: Pool Integration Registry	(pool addresses)
A6: LP State	(collateral mint, seeded flag, volume)
A7: Collateral Vault	(SPL TokenAccount, internal AMM reserve)

Contract B (Math Contract):

B1: Severity State Account	(s, previous_s, observation window)
B2: Ladder State Account	(r, tau, permutations_tried)
B3: Intensity State Account	(I_1 through I_8 array)
B4: Active Set State	(current rung ordering for §2.9)
B5: Parameter Account	(delta, alpha, lambda, etc.)

Total: twelve PDAs (the original whitepaper specified ten; A6, A7, B4 were added for the internal AMM and the §2.9 active-set storage). Each PDA is derived deterministically from a seed string and the program ID. No private keys. No discoverable addresses.

3.4 Concurrency Property

Most user transactions do not write to Contract B’s ladder or intensity accounts (they only update severity in B1). This means **Nyxanic naturally creates a fast path during calm and a slow path during stress**. High-volume operation parallelizes; rare ladder transitions serialize.

4 The Heartbeat Protocol

4.1 The `evaluate_transaction` Instruction

Per user transaction, Contract A invokes Contract B with a small payload:

```
fn evaluate_transaction(  
    current_price: u64,    // fixed-point, scaled 1e6  
    direction_hint: i8,    // +1, -1, or 0  
    current_slot: u64,  
)
```

4.2 The Directive Response

Contract B returns the directive via Solana's `set_return_data`:

```
struct DirectiveResponse {  
    execution_mode:      u8,    // EXECUTE | THROTTLE | ROUTE |  
                           HALT  
    fee_adjustment:      i64,    // delta from base fee, signed  
        by direction  
    spread_adjustment:    u64,  
    throttle_factor:      u16,    // fixed-point 0-10000 (10000 =  
        no throttle)  
    mint_burn_delta:      i64,    // negative = burn from treasury  
    collateral_ratio_target: u32,    // 0 = no constraint  
}
```

The directive is layered: lighter rungs occupy fewer fields; heavier rungs add more. The default directive (EXECUTE, all zeros except `throttle_factor` = 10,000) corresponds to the resting state.

4.3 Defensive Fallback

If Contract B fails (panic, malformed return, invalid directive), Contract A applies the default directive (EXECUTE, no adjustments) and emits a **BrainFailure** event with a numeric reason code (1 = parse error, 2 = no return data, 3 = CPI failure). The body continues operating at minimum capacity even with a temporarily confused brain.

5 Self-Sustainability

Nyxanic feeds on transaction flow itself. It generates the energy required to defend the peg from the very disturbances threatening it.

Fee Metabolism. Every swap mints the extracted spread+fee portion as the stablecoin into the Fee Collector treasury. Higher volume means more energy.

Spread Harvesting. Widening spreads during volatility extracts the difference, profiting from the disturbances that threaten stability.

Friction Revenue. Dynamic fees on destabilizing transactions generate income precisely when Nyxanic is under stress.

Treasury-Funded Supply Contraction. When the brain demands supply contraction (rung ℓ_6 , `mint_burn_delta` < 0), the body autonomously burns from its accumulated trea-

surey via the permissionless `rebalance_supply` instruction. No autonomous mint occurs (positive deltas emit a signal event only, never dilute).

The principle:

Stress produces energy. Nyxanic strengthens under attack.

6 Birth Sequence

Nyxanic is initialized in strict steps, in dependency order:

1. Deploy Programs A and B (program IDs hardcoded in each other's source via `declare_id!`)
2. Genesis authority (deployer key) ready
3. Initialize Parameter Account B5 (with parameter validation: $\tau_{\text{cool}} > \tau_{\text{wait}}$, θ_r strictly increasing, etc.)
4. Initialize Brain State Accounts B1, B2, B3, B4 (genesis: all dormant)
5. Initialize Mint A1 and Reserve Vault A2 (initial supply: zero)
6. Initialize Body State A3 and Fee Collector A4
7. Initialize Pool Registry A5
8. Initialize LP state A6 and Collateral Vault A7 (registers the chosen collateral mint)
9. Authority handoff via `seal_body` (deployer's pubkey on `body_state` and `pool_registry` replaced by `Pubkey::default()`; `apply_directive` and `register_pool` permanently fail thereafter)
10. First breath via `first_breath` (initial liquidity provider deposits seed collateral; matching stablecoin minted to them; `lp_state.seeded` set true; one-shot)

After step 9, the deployer key has no special power over Nyxanic. The system is fully trustless from this point forward.

7 Properties

7.1 Capital Efficiency

Nyxanic never deploys capital to defend its peg under normal operation. Reserves remain idle. Capital is consumed only at the heaviest rungs (ℓ_5 through ℓ_7), and even then only to the minimum extent required.

7.2 Cost Frugality

- **Constant memory:** two prices, ladder position, eight intensity values, eight active-set bytes, bounded permutation count.
- **O(1) operations:** closed-form Hill function via fixed-point integer arithmetic, conditional ladder transitions, single-multiplication intensity updates.
- **Self-erasing state:** permutation count resets when severity exits the medium band; no accumulating storage.

7.3 Adversarial Resistance

The deterministic permutation at medium severity prevents adversaries from predicting which organ will lead the response. The hysteresis gap ($\tau_{\text{cool}} > \tau_{\text{wait}}$) prevents oscillation attacks. The terminal stop-flag is the unconditional defense.

7.4 Trustlessness

After birth, no key controls Nyxanic. All accounts are PDAs. All authority is held by program-derived addresses. The deployer is no longer privileged. Setting the program upgrade authority to `None` (via `solana program set-upgrade-authority --final`) freezes the bytecode itself — not done in the reference deployment to allow continued iteration on devnet.

7.5 Determinism

All mathematics is computed in fixed-point integer arithmetic. No floating point. No non-determinism across hardware. Every Solana validator computes identical results. The permutation seed is derived from public on-chain state, so every observer reaches the same shuffle.

7.6 Observability

Nyxanic’s nervous activity is publicly visible through Anchor events emitted on every transaction (`HeartbeatEvent`, `LpSwapEvent`, `SupplyContracted`, `BrainFailure`, etc.). External monitors can reconstruct full state without reading any account directly.

7.7 Trustless Price Source

The reference implementation derives the observed price internally from the AMM’s own state:

$$P(t) = \frac{\text{vault balance (collateral units)} \cdot \text{PRICE_SCALE}}{\text{stablecoin supply}} \quad (11)$$

The instruction `heartbeat_observed` takes only a slot, never a price. No caller can fake the price input. This eliminates the oracle as an attack surface for the closed-loop case where the AMM is the only market.

8 Domain Independence

The mathematical architecture is defined entirely in terms of:

- An observable quantity $P(t)$ and a target P^*
- An ordered set of controllable variables $\mathcal{L}$
- A severity signal $s(t)$ derived from error $e(t)$
- Time as the sole governing factor

By substituting appropriate definitions, the architecture maps to:

- Stablecoin peg maintenance (this paper)
- Temperature regulation in physical systems
- Blood pressure regulation in biological organisms

- Traffic flow management in networks
- Voltage stabilization in electrical grids
- Any system requiring dynamic equilibrium under disturbance

The mathematics does not change. Only the variables.

9 Reference Implementation

The repository at <https://github.com/morphym/nyxanic> contains a complete reference implementation of the architecture described in Sections 2–4. The implementation is divided into a pure-math library and three Solana programs.

9.1 Repository Layout

```
nyxanic/
  sim/                                # math library + offline
    simulator
    src/lib.rs                        # organism-math (no_std, pure
      functions)
    src/main.rs, src/experiments.rs # offline experiments

  organism/                          # Anchor workspace
    programs/
      organism-math/    -> renamed crate `organism-brain` (Contract B
        )
      organism-token/  -> crate `organism-token`           (Contract A
        )
      nyxanic-test-token/ -> crate `nyxanic-test-token`    (NTT,
        devnet)

  tests/                             # mocha + ts-mocha integration tests
  scripts/terminal.ts # interactive TUI
```

9.2 Math Library (organism-math)

The math is a `no_std` Rust crate with no dependencies. Single-file public API:

```
pub fn compute_severity(price: u64, params: &Params) -> u64;
pub fn step_ladder(state: &mut BrainState, params: &Params);
pub fn step_intensity(state: &mut BrainState, params: &Params);
pub fn step_permutation(state: &mut BrainState, params: &Params, slot
: u64);
pub fn evaluate(
  state: &mut BrainState,
  price: u64,
  direction: i8,
  slot: u64,
  params: &Params,
) -> Directive;
pub fn compute_fee(s: u64, direction: i8, params: &Params) -> i64;
pub fn compute_direction(current_price: u64, tx_effect: i64) -> i8;
```

The on-chain brain crate (**organism-brain**) wraps these functions; it does not reimplement the math. Convenient byproduct: any consumer of **organism-math** computes identically to the chain.

9.3 Body — Token Contract (**organism-token**)

Instructions implemented:

- **initialize_mint**, **initialize_body** — birth steps 5–7
- **initialize_lp** — registers a collateral mint and creates the collateral vault (birth step 7 cont.)
- **first_breath** — one-shot seed of initial liquidity (birth step 10)
- **seal_body** — one-shot authority handoff (birth step 9)
- **heartbeat** — caller-supplied price; CPIs into brain, stores returned directive
- **heartbeat_observed** — price derived from on-chain state; trustless
- **lp_swap_in** — collateral $\rightarrow$ stablecoin; honors halt, throttle, spread, fee, ℓ_7 ratio constraint
- **lp_swap_out** — stablecoin $\rightarrow$ collateral; same gating
- **rebalance_supply** — ℓ_6 implementation; permissionless trigger that burns from **fee_collector** when **mint_burn_delta** < 0
- **mint_tokens**, **burn_tokens**, **transfer_tokens** — raw SPL operations also gated by stored directive (halt, fee, throttle)
- **register_pool** — admin-only (ℓ_5 stub; sealed at birth step 9)
- **apply_directive** — admin-only test path; sealed in production

9.4 Brain — Math Contract (**organism-brain**)

Instructions:

- **initialize_params** — B5 init with validation
- **initialize_brain** — B1, B2, B3 init (genesis dormant)
- **initialize_active_set** — B4 init with order [1, 2, 3, 4, 5, 6, 7, 8]
- **evaluate_transaction** — the heartbeat. Reads B1–B5, calls **organism_math::evaluate**, writes back, returns serialized **DirectiveResponse** via **set_return_data**.

9.5 Implementation Notes Beyond the Original Specification

The reference implementation extends the original whitepaper in five places:

1. **Descent rule relaxed.** Strict $\Delta s < 0$ deadlocks the ladder when severity reaches and stays at zero (a true equilibrium produces $\Delta s = 0$). The implementation uses $\Delta s \leq 0$, so the ladder drains correctly after full recovery. Verified on devnet (Section 11).
2. **Internal AMM.** The original spec leaves liquidity to external pools. The reference implementation builds a self-contained constant-ratio AMM. Vault holds collateral; supply tracks the mint. Bidirectional swaps apply the directive’s fee/spread; extracted value is

minted as stablecoin into the Fee Collector treasury (so total supply = vault collateral by construction; see also Section 11.5 for the asymmetric recovery limitation).

3. **Trustless price oracle.** `heartbeat_observed` computes $P(t) = \text{vault}/\text{supply}$ on-chain. No caller-supplied price.
4. **Active-set as a PDA.** The §2.9 history $\mathcal{H}(t)$ is replaced by a permutation count and the current ordering, stored in account B4. Permutations exhaust at $r!$ and force ladder advance. The Fisher–Yates seed is documented in §2.9.
5. **Asymmetric supply correction.** `rebalance_supply` acts only on `mint_burn_delta < 0` (burn from treasury, safe). Positive deltas emit a `SupplyExpansionRequested` event for off-chain market response; the body never autonomously mints unbacked supply.

10 Measured Compute Cost

The original whitepaper estimated $\sim 5,250$ CU per heartbeat. Direct measurement on devnet against the deployed brain (program ID in Appendix C) yields the following totals *including all Anchor framework overhead* (account deserialization, PDA seed validation, Rent sysvar reads, return-data encoding):

Scenario (<code>heartbeat_observed</code> via CPI)	CU consumed	% of 1.4M budget
At peg (resting, mode = EXECUTE)	25,152	1.80%
0.5% depeg (mild)	25,163	1.80%
1% depeg (moderate)	25,160	1.80%
5% depeg (severe)	25,349	1.81%
10% depeg (extreme)	25,349	1.81%
30% depeg (catastrophic)	25,349	1.81%
Recovery	25,152	1.80%
Sustained attack (rung climb)	25,358–25,362	1.81%

Observations:

- **Cost is essentially flat across severity.** The Hill function, ladder transition, intensity update, and permutation step are all $O(1)$ over fixed $N = 8$ rungs. CU varies by under 1% between full rest and catastrophic depeg.
- **Anchor overhead dominates.** The pure math (Hill, ladder, intensity, permutation, directive build) is $\sim 5,000$ CU — close to the original estimate. The remaining $\sim 20,000$ CU is Anchor framework: PDA validation across five brain accounts, borsh deserialization, and CPI return-data setup.
- **Still well within budget.** Even with framework overhead, the heartbeat consumes $< 2\%$ of Solana’s 1.4M CU per-transaction limit. A swap-with-heartbeat ($\sim 25\text{k}$ brain CU + $\sim 10\text{k}$ SPL operations) lands near 35k CU — 2.5%.

11 Observations from Devnet

The reference deployment was exercised through twelve integration test scenarios. Results below are direct quotes from devnet runs.

11.1 Severity (§2.3)

Hill function output across price deviations, $\delta = 0.005$, $\alpha = 3$:

price	error	severity s
1.0000	0.0000	0.0000
0.9990	0.0010	0.0079
0.9980	0.0020	0.0602
0.9950	0.0050	0.5000
0.9900	0.0100	0.8889
0.9800	0.0200	0.9846
0.9500	0.0500	0.9990
0.9000	0.1000	0.9999

Half-activation at $|e| = \delta$ holds to four decimal places.

11.2 Hysteresis (§2.7)

Oscillating price between 0.995 and 1.005 for 100 ticks at the rung-1 threshold, after warming the system to rung 4: **0 direction reversals**. The asymmetry $\tau_{\text{cool}} = 30 > 10 = \tau_{\text{wait}}$ holds rung position despite continuous flapping at the boundary.

11.3 Permutation (§2.9)

Driving severity into the medium band ($\theta_{\text{low}} = 0.2$, $\theta_{\text{high}} = 0.6$) with $\Delta s_{\text{obs}} \geq 0$ at rung 6 produces, on devnet:

```
[0] [1,2,3,4,5,6,7,8]
[1] [5,1,6,3,4,2,7,8]
[2] [3,6,2,4,1,5,7,8]
```

Active rungs $\{1, \dots, 6\}$ permute. Tail $\{7, 8\}$ stays in place because they are not active. Counter resets to 0 when severity exits the band.

11.4 6 Supply Correction

Forced above-peg state by donating 50,000 NTT collateral directly into the vault (price spike to 1.455). Brain climbs to rung 7 within ~ 30 heartbeats, `mint_burn_delta` becomes $-93,530$. Calling `rebalance_supply` burns exactly 93,530 stablecoin from the Fee Collector. Treasury balance drops by the same amount; total supply drops by the same amount. Directive's `mint_burn_delta` is zeroed (applied flag).

11.5 7 Collateral Ratio Enforcement

During ladder descent through rung 6 (mode = ROUTE), the directive carries `collateral_ratio_target = 2.0000`. Vault/supply at the moment is 1.4550. A `swap_in` attempt is rejected with `InsufficientCollateral` (the post-swap ratio would dilute below 2.0). `Swap_outs` at the same moment are still permitted because they improve the ratio. 7 is therefore a *soft brake* — distinct from ℓ_8 (Halt), which is unconditional.

11.6 Authority Handoff (Step 9)

After `seal_body`: `body_state.authority` and `pool_registry.authority` become `Pubkey::default()` (11111111...). `apply_directive` returns `Unauthorized` to the original deployer. `register_pool` returns `Unauthorized`. `heartbeat_observed` continues to update the body's directive normally (no authority gate). Re-sealing also fails: the deployer no longer matches the (now-default) authority.

11.7 Asymmetric Recovery (Limitation)

The internal AMM cannot self-correct an above-peg state. When external collateral is donated to the vault (a benign attack), vault grows but supply does not, the brain detects the deviation and halts trades, and there is no autonomous mechanism to drain the excess. `swap_outs` improve the ratio but require burning supply, which is bounded by user holdings. `swap_ins` are halted. The system is self-stable but not self-healing in this direction. A future `claim_excess` instruction (mint backed against the over-collateralization, distributed permissionlessly) would close this gap; not implemented in v1.1.

12 Test Currency: NTT (Nyxanic Test Token)

For devnet experimentation, a separate program `nyxanic-test-token` provides a permissionless, mintable SPL token used as the LP's collateral. Production deployments would substitute USDC or another externally-issued asset.

12.1 Hard Constraints

Property	Value
Total supply cap	1,000,000,000 NTT
Reserved for liquidity (one-shot <code>seed_liquidity</code>)	90% (900M)
Available via airdrop	10% (100M)
Max per airdrop call	50,000 NTT
Cooldown per wallet	8 hours
Decimals	6

12.2 Mechanism

Airdrop is keyed by the recipient *wallet* (not the token account) via a per-wallet `AirdropRecord` PDA seeded by the wallet pubkey. Creating a new associated token account does not bypass the cooldown. Total airdrop quota is hard-capped at 10% of supply — once exhausted, no further airdrops are possible.

`seed_liquidity` is a separate one-shot instruction sized for bootstrap deposits into the AMM. It checks the LP cap (90%) but is otherwise permissionless: anyone may seed.

12.3 Errors

```
NttError::InvalidAmount          // amount = 0 or > 50,000
NttError::AirdropOnCooldown      // wallet's last airdrop < 8h ago
NttError::AirdropQuotaExhausted  // would exceed 10% global cap
NttError::LiquidityQuotaExhausted // would exceed 90% global cap
NttError::WrongMint              // token account is for a
    different mint
NttError::WalletMismatch         // recipient.owner !=
    recipient_wallet
```

13 Tooling

13.1 Random Multi-Account Test

`tests/random.ts` spawns seven keypairs, funds each with 0.01 SOL, airdrops 600 NTT each (4,200 NTT total). The loop performs random buys and sells from random wallets at random fractions of balance, refreshing the brain via `heartbeat_observed` before each iteration. A live ASCII dashboard renders supply, vault, derived price, fees, brain rung/mode/fee/spread/throttle, per-wallet balances, and recent activity. Loop terminates on any non-EXECUTE mode (throttle, route, halt) or after a fixed iteration limit.

13.2 Interactive Terminal

`scripts/terminal.ts` is a readline-based TUI for direct human interaction:

```
> b   buy NYX with NTT (prompts for NTT amount)
> s   sell NYX for NTT (prompts for NYX amount)
> a   airdrop NTT to self (50k max, 8h cooldown enforced)
> h   heartbeat (refresh brain directive)
> r   re-render dashboard
> q   quit
```

The dashboard shows organism state, brain state, the user's wallet (SOL/NTT/NYX balances), airdrop cooldown remaining, and global NTT supply counters. All Anchor errors are rendered with their human-readable name (`Halted`, `Throttled`, `InsufficientCollateralRatio`, `AirdropOnCooldown`, etc.).

14 Conclusion

Nyxanic is a stablecoin not because it holds tokens. It is a self-sustaining system because it senses, decides, acts, and rests — governed entirely by Control Theory, on-chain, atomically, deterministically, and frugally.

It defends a hard \$1.00 peg without spending capital. It feeds on the flow that passes through it. It strengthens under attack. It uses time as its only judge.

The architecture is incomplete by design — complexity is high. But it handles emergent complexity perfectly through small, well-tuned mathematical primitives composing into vast behavioral capability.

Small mathematics. Vast emergent capability. Constant cost.

This is Nyxanic.

A Glossary of Symbols

Symbol	Meaning
$P(t)$	Observed price at time t
P^*	Target peg (\$1.00)
$e(t)$	Error: $P^* - P(t)$
$s(t)$	Severity in $[0, 1]$
$\Delta s(t)$	Severity velocity
d	Transaction direction $\in \{-1, +1\}$
$f(t)$	Dynamic fee
$r(t)$	Ladder position $\in \{0, \dots, 8\}$
$\mathcal{L}$	Ordered set of controllable variables
$\mathcal{A}(t)$	Currently active set (rung ordering)
Π	Deterministic permutation operator
$I_r(t)$	Intensity of rung r , $\in [0, 1]$
$\tau(t)$	Time on current rung
θ_r	Severity threshold for rung r
Δs_{obs}	Severity change over τ_{obs} window

B Compute Cost Comparison (Measured)

Operation	Compute Units	% of Budget
Solana transaction budget	1,400,000	100%
SPL token transfer (baseline)	$\sim 5,000$	0.36%
Heartbeat resting (whitepaper estimate)	$\sim 5,250$	0.38%
Heartbeat resting (measured)	25,152	1.80%
Heartbeat full activation (measured)	25,362	1.81%
Combined swap + heartbeat (measured)	$\sim 35,000$	2.50%

The pure math layer matches the original estimate ($\sim 5,000$ CU). Anchor framework overhead — PDA validation, borsh serialization, CPI return-data plumbing — adds a roughly constant 20,000 CU. Even so, the system consumes well under 3% of the per-transaction budget at all severity levels.

C Deployed Addresses (Solana Devnet)

Component	Address
Brain program	AFQ8VKgobymzFYCv4NpcSZRpjVyjW9o1uYusRsrgdcqk
Body program	fvmZ3pBDXY9H3a4ViK5u9e3kPXC9abpADqA3N9LcVb
NTT program	CwgFaQX7APeKvLGiJAxLUGCReooTeM3YzzocQ6bvNYow
NTT mint (PDA)	GFZrayFQSExDD5B7Jquk69v72j87nWoZ5qTxho9R1KNF
NYX mint (PDA)	FLoyCdnSFtppF9Q64xNP462HPKhTcw54yR3jMKkzborh

Repository: <https://github.com/morphym/nyxanic>