TECHNICAL WHITEPAPER • PROTOCOL ARCHITECTURE

Junk World Assets (JUNK) Protocol

A Permissionless Escrow and Weighted Random Acquisition Protocol with Configurable $JUNK Incentives

AUTHOR / ORGANIZATION

Junk World Assets Architecture Team

PUBLICATION DATE

August 2026

SPECIFICATION VERSION

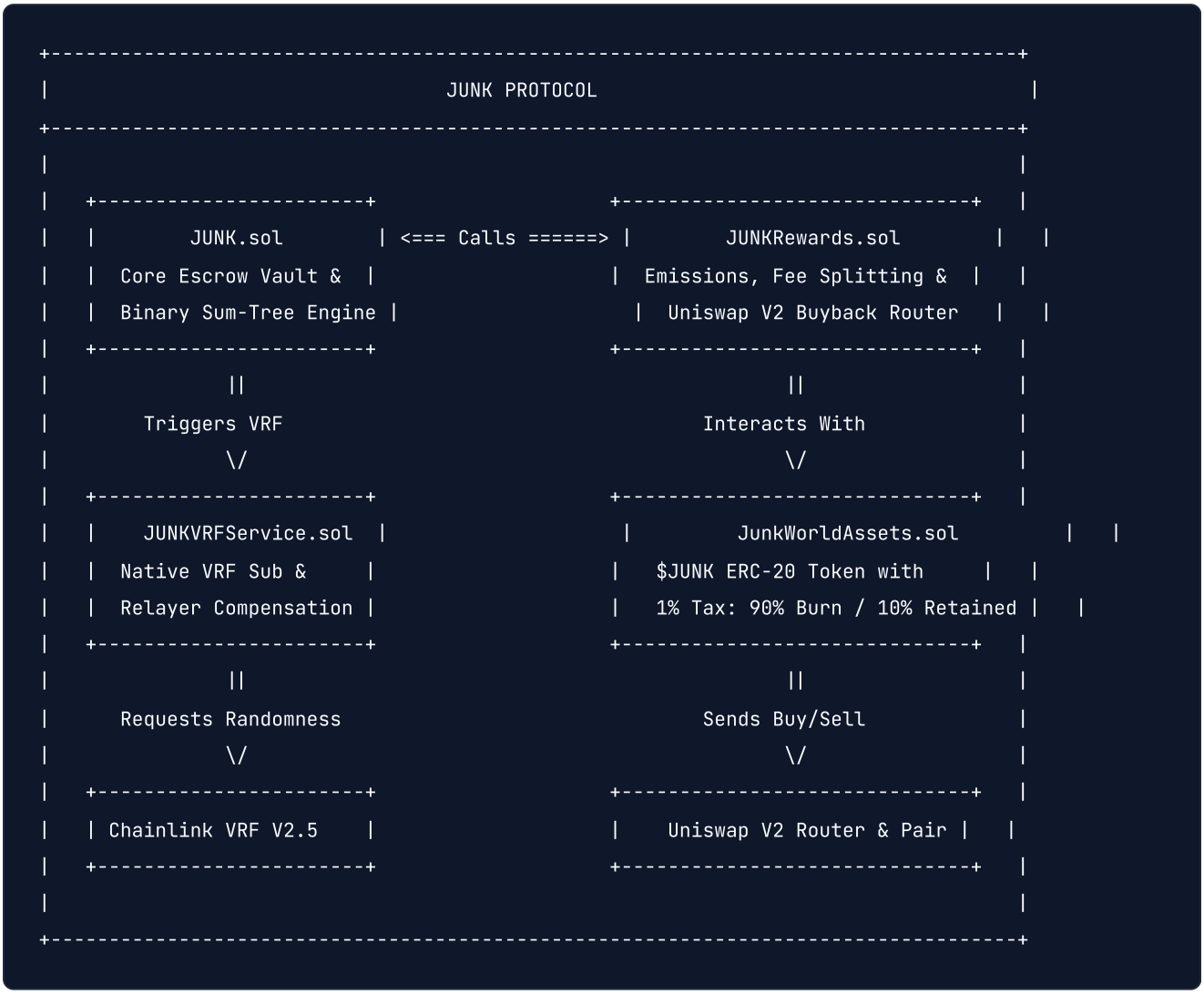
v1.1.0

Abstract: The Junk World Assets (JUNK) Protocol is an on-chain position escrow and randomized acquisition system deployed on Ethereum mainnet, with launch preparation ongoing. Depositors lock ERC-721 NFTs or ERC-20 token amounts into a shared escrow vault backed by committed ETH. Position weights are inversely proportional to backing, producing an integer approximation of harmonic-mean pool pricing. Chainlink VRF V2.5 makes the draw result unpredictable before fulfillment, reducing selection manipulation but not eliminating every form of MEV. Configurable acquisition and settlement paths can buy and burn **$JUNK**. On taxable Uniswap V2 buys and sells, the token charges 1%; 90% of that fee is sent to the dead address and 10% is retained for eventual treasury conversion. This document describes the implemented contract behavior, the deployed configuration, and its operational limits.

Table of Contents

1. System Architecture Overview

The Junk World Assets consists of four primary smart contract modules designed with strict single-responsibility principles, isolated storage domains, and gas-optimized Solady primitives:



Core Subsystems Matrix

Contract	Core Responsibility	Key Dependencies	Solady Utilities Used
`JUNK.sol`	Asset escrow, binary sum-tree selection, VRF settlement dispatch, listing lifecycle, standing bid settlement	`JUNKRewards`, `JUNKVRFService`, `VRFCoordinatorV2Plus`	`ERC721`, `ERC20`, `Ownable`, `ReentrancyGuard`, `SafeTransferLib`
`JunkWorldAssets.sol`	Protocol incentive token ($JUNK, 1B initial supply), 1% V2 buy/sell tax, 90% fee-share burn, 10% fee-share treasury retention, visible dead-address burn function	`IUniswapV2Router02`, `IUniswapV2Pair`	`ERC20`, `Ownable`, `ReentrancyGuard`
`JUNKRewards.sol`	Seven-day fixed-rate emissions, dynamic acquisition allocation, user allowance tracking, V2 token swaps	`JunkWorldAssets`, `IUniswapV2Router02`	`Ownable`, `ReentrancyGuard`, `SafeTransferLib`, `FixedPointMathLib`
`JUNKVRFService.sol`	Purchaser VRF fee escrow, native Chainlink subscription top-up, relayer gas reimbursement	`IVRFCoordinatorV2Plus`, `JUNK`	`Ownable`, `ReentrancyGuard`, `SafeTransferLib`, `FixedPointMathLib`

JUNK Deployment Status

Core deployed; launch preparation in progress. As of September 20, 2026, the four Uniswap V2 JUNK contracts are deployed on Ethereum mainnet and exact-verified on Etherscan. The fresh Chainlink VRF subscription is configured with both consumers and funded with 0.155 ETH. Trading and acquisitions remain disabled.

- **JunkWorldAssets**: `0xa88fc01053f0c28dfadafb05c4705fa7c2ed7954`
- **JUNKRewards**: `0xcb4ef27f26808c23e94592b2da642f5ead56052d`
- **JUNKVRFService**: `0x2c0bac7c9e1caa3401fbe4377b1d8d0028e2ee35`
- **JUNK**: `0xa4acf991777099804631b274b76639ca081c23a8`

The core deployment transferred 350,000,000 $JUNK to rewards (175,000,000 each for depositors and purchasers) and 650,000,000 JUNK to the protocol recipient. Fee-vault deployment is reserved for manual execution; liquidity and protocol launch are separate pending steps.

---

2. Core Protocol Mechanics & Mathematical Model

2.1 Permissionless Position Escrow

The Junk World Assets operates a permissionless floor vault for compatible contracts. Depositors lock an asset—either an ERC-721 NFT or an ERC-20 token amount—accompanied by an ETH deposit known as **Backing** (V_i). Listings must pass the contract's collection checks, transfer successfully, and meet the minimum backing (0.01 ETH by default).

$$\textbf{Minimum Backing Requirement: } V_i \geq V_{\text{min}} = 0.01 \text{ ETH}$$

When depositing, the committed ETH backing stays in contract escrow and funds the settlement alternatives available after selection. It is not an unconditional depositor right to reclaim the position at any time: the purchaser controls the settlement choice during the 24-hour window, after which the depositor has two reclaim paths with different payouts.

2.2 Inverse Selection Weighting

To align economic incentives and ensure low-backed positions carry a proportional chance of being drawn while high-backed positions act as high-value anchors, selection probability is inversely proportional to position backing V_i.

For a listing i with ETH backing V_i, its integer selection weight W_i is computed as:

$$W_i = \left\lfloor \frac{10^{36}}{V_i} \right\rfloor$$

Key Implications:

- **Low Backing ($V_i \downarrow$)**: Higher selection weight ($W_i \uparrow$), resulting in a higher probability of selection during an acquisition pull.
- **High Backing ($V_i \uparrow$)**: Lower selection weight ($W_i \downarrow$), resulting in a lower probability of selection.

2.3 Harmonic Mean EV & Acquisition Pricing

The acquisition price for any pool pull is the contract's current integer-weighted EV of active positions plus the owner-configured surcharge (10% by default).

The total pool selection weight W_{total} and weighted backing total V_{weighted} are defined as:

$$W_{\text{total}} = \sum_{i=1}^{N} W_i$$

$$V_{\text{weighted}} = \sum_{i=1}^{N} (W_i \cdot V_i)$$

The Pool Expected Value (EV) represents the expected backing of a randomly drawn position:

$$\text{EV} = \frac{V_{\text{weighted}}}{W_{\text{total}}} = \frac{\sum_{i=1}^{N} (W_i \cdot V_i)}{\sum_{i=1}^{N} W_i}$$

In an idealized real-number model, substituting $W_i = \frac{\Xi}{V_i}$ (where $\Xi = 10^{36}$) gives:

$$\mathbf{EV} = \frac{\sum_{i=1}^{N} \left(\frac{\Xi}{V_i} \cdot V_i \right)}{\sum_{i=1}^{N} \frac{\Xi}{V_i}} = \frac{N \cdot \Xi}{\Xi \sum_{i=1}^{N} \frac{1}{V_i}} = \frac{N}{\sum_{i=1}^{N} \frac{1}{V_i}} = H(V_1, V_2, \ldots, V_N)$$

This is the harmonic mean in the idealized model. The deployed code uses $W_i = \lfloor \Xi / V_i \rfloor$ and integer division for `weightedBackingTotal / totalWeight`, so the live EV is a truncated, integer-weighted approximation rather than an exact harmonic mean.

Acquisition Fee Equation:

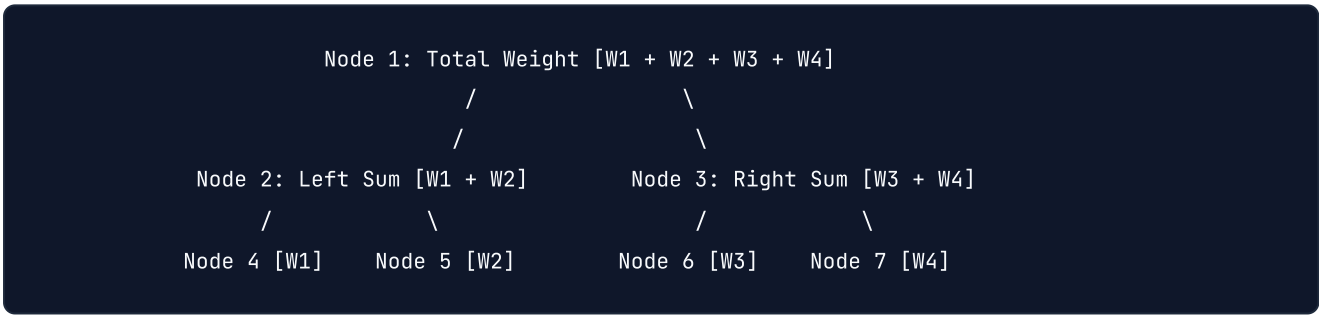
The gross ETH required from a purchaser to trigger an acquisition pull is, at the default 10% surcharge:

$$P_{\text{acq}} = \mathbf{EV} \cdot \left(1 + \frac{\gamma_{\text{surcharge}}}{10000} \right) = \mathbf{EV} \cdot 1.10$$

where $\gamma_{\text{surcharge}} = 1000$ bps (10%).

2.4 $O(\log N)$ Binary Sum-Tree Engine

To maintain scalable on-chain selection without iterating over arrays, `JUNK.sol` implements a 32-depth static Binary Sum-Tree structure.



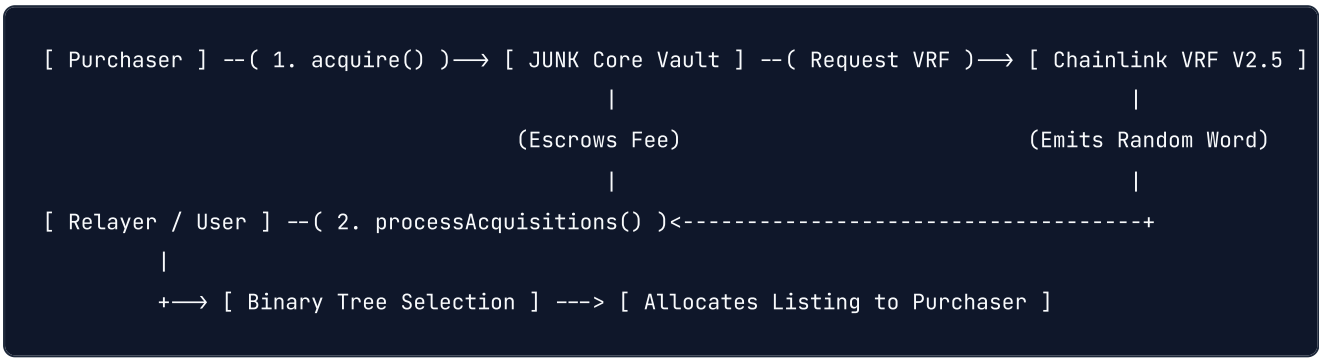
- **Capacity**: $2^{32} \approx 4.29 \times 10^9$ slots.
- **Leaf Base**: $\mathbf{LEAF\_BASE} = 2^{32}$. Slot s corresponds to tree node $\mathbf{LEAF\_BASE} + s - 1$.
- **Updates ($O(\log N)$)**: Modifying a listing's backing updates leaf weight and bubbles the delta up to root node 1.
- **Selection Traversal ($O(\log N)$)**:
  Given a pseudorandom number $R = \mathbf{VRF\_Word} \pmod{W_{\text{total}}}$:
  1. Start at root `node = 1`.
  2. Compare R against `tree[left_child]`.
  3. If $R < \mathbf{tree}[left\_child]$, traverse to left child (`node = left_child`).
  4. Else, set $R \leftarrow R - \mathbf{tree}[left\_child]$ and traverse to right child (`node = left_child + 1`).
  5. Repeat until reaching a leaf node (`node ⩾ CAPACITY`).

---

3. Acquisition Pipeline & Chainlink VRF V2.5

3.1 Asynchronous Two-Phase Acquisition

To make the draw result unavailable at request time, acquisition settlement is split into request and fulfillment/processing phases. VRF does not by itself eliminate all front-running, block reorganization, or validator/MEV risks:



1. **Request Phase (`acquireWithSlippage`)**:
   - Purchaser deposits gross acquisition fee P_{acq} plus VRF service fee P_{vrf}.
   - Protocol assigns an incremental sequence number S, calculates deadline block ($B_{\text{curr}} + 30$), and submits a random word request to Chainlink VRF V2.5.
   - Acquisition status is marked `Pending`.
2. **Fulfillment & Processing Phase (`processAcquisitions`)**:
   - Chainlink VRF coordinator calls `rawFulfillRandomWords`, caching the random word and marking status as `Ready`.
   - The VRF callback may attempt an internal processing call when gas permits; otherwise anyone may call `JUNK.processAcquisitions(maxCount)`. The optional `JUNKVRFService` wrapper is restricted to configured operators and can reimburse processing gas within its caps.
   - The contract executes binary tree selection, updates listing status to `Allocated`, sets `allocatedAt = block.timestamp`, and triggers surcharge fee distribution.

3.2 Autonomous VRF Service & Relayer Buffer

`JUNKVRFService.sol` provides an optional operator-gated processing wrapper and VRF subscription funding path; it does not guarantee seamless settlement or remove the need for a separate processing transaction.

- **Purchaser VRF Fee**:

$$P_{\text{vrf}} = \text{serviceGasEstimate} \cdot P_{\text{gas}} \cdot \left(1 + \frac{\text{serviceMarginBps}}{10000}\right) + \text{flatWei}$$

(Defaults: $\text{gasEstimate} = 800,000$, $\text{margin} = 30\%$*).*

- **Subscription Auto-Funding**:
  `JUNKVRFService` checks the Chainlink VRF subscription native ETH balance during request preparation and operator/top-up calls. If the balance falls below the required threshold:

$$\text{Required Balance} = \text{buffer} + (N_{\text{unfulfilled}} + N_{\text{new}}) \cdot \text{maxFulfillmentCost}$$

  The service contract calls `fundSubscriptionWithNative` as part of those calls, subject to its available balance.

- **Relayer Compensation**:
  Configured operators calling the service wrapper may receive capped gas reimbursement from `JUNKVRFService`:

$$\text{Reimbursement} = \min\left(\text{GasUsed} \cdot \min(P_{\text{gas}}, P_{\text{cap}}), \text{maxReimbursementWei}\right)$$

3.3 Slippage Protection & Timeout Management

- **Price Slippage Tolerance**: Purchases record a user-supplied negative-slippage tolerance and use the protocol's positive-slippage tolerance. During processing, the acquisition is refunded if the live fee moves outside either bound around the escrowed fee; the refund is credited to `acquisitionRefundCredit` for later withdrawal.
- **VRF Timeout**: If Chainlink VRF does not fulfill within `selectionTimeoutBlocks` (default 30 blocks), the request transitions to `Expired` and full refunds are unlocked.

---

4. Settlement Engine & Economic Flows

4.1 Purchaser Settlement Choices

When a position i with backing V_i is allocated to a purchaser, the purchaser may act during the default **24-hour Settlement Window** (`settlementWindow = 86400s`). The owner can configure this window subject to the contract's bounds. The purchaser has three mutually exclusive resolution choices:

Choice 1: Keep Asset (`keepNFT`)

- **Purchaser**: Takes ownership of the NFT or Token Bag.
- **Depositor**: Receives net backing payout ($V_i \cdot (1 - \text{ownerSettlementFeeBps})$).
- **Protocol Fee**: Protocol takes 1% fee ($\text{ownerSettlementFeeBps} = 100$ bps), swapped into $JUNK$ for the protocol treasury.

Choice 2: Accept Standing Bid in ETH (`acceptDepositorBid`)

- **Purchaser**: Surrenders the asset back to the depositor and receives **85% of backing in ETH**:

$$\text{Payout}_{\text{ETH}} = V_i \cdot 0.85$$

- **15% Discount Burn**: The remaining 15% of backing ($V_i \cdot 0.15$) is sent through `JUNKRewards.buyFor` and, when the configured swap succeeds, the resulting $JUNK$ is sent to `DEAD_ADDRESS` (`0x...dEaD`). Swap failure is caught by the core contract, so this is an attempted burn path rather than an unconditional guarantee.
- **Depositor**: Recovers their original NFT or Token Bag.

Choice 3: Accept Standing Bid in $JUNK$ (`acceptBidAsTokens`)

- **Purchaser**: Surrenders the asset to the depositor and receives 85% of backing converted directly into $JUNK$ **tokens** via `JUNKRewards.buyFor`.
- **15% Discount Burn**: The 15% discount surrendered ($V_i \cdot 0.15$) follows the same configured V2 buy-and-burn path.
- **Depositor**: Recovers their original NFT or Token Bag.

Depositor Reclaim (Post-Expiration Guard)

If the purchaser takes no action within 24 hours, the settlement lock expires:

- Depositor may call `depositorReclaimBacking` (recovers backing minus 1% protocol fee; NFT delivered to purchaser).
- Depositor may call `depositorReclaimNFT` (pays purchaser 85% payout; recovers NFT; 15% discount burned).

4.2 The 15% Settlement Discount Burn

The 15% settlement discount is an implemented burn path when the rewards module, router, and liquidity are configured and the buy succeeds. The core then calls `IJunkWorldAssets.burn()`, which transfers the purchased tokens to the visible dead address. This reduces circulating availability; it does not reduce the ERC-20 `totalSupply` value because the token's burn function is a dead-address transfer.

$$\textbf{ETH Swapped \& Burned} = V_i \cdot 0.15$$

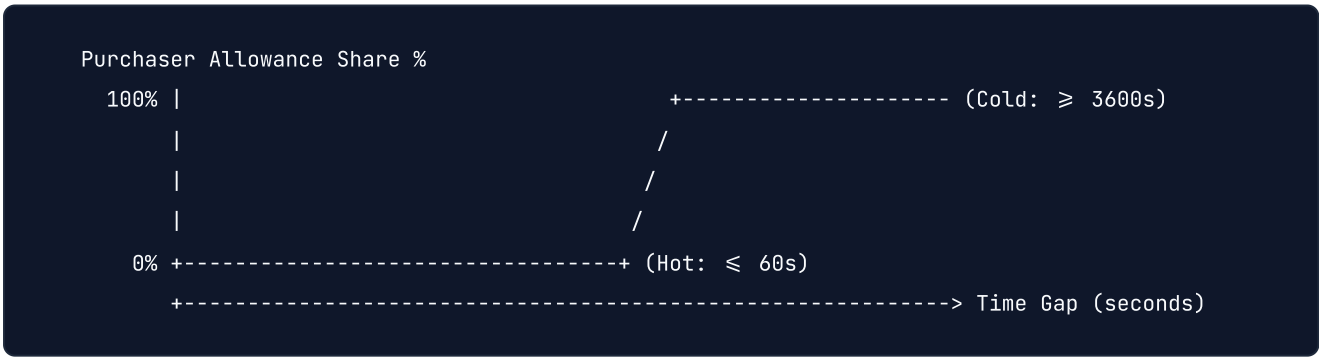
$$\textbf{Tokens Burned} = \textbf{UniswapV2\_Swap}\left(V_i \cdot 0.15\ \text{ETH} \longrightarrow \$JUNK\right) \longrightarrow \textbf{DEAD\_ADDRESS}$$

4.3 Dynamic Surcharge Split & Hot/Cold Curve

The protocol derives a purchaser allocation from the 10% acquisition surcharge ($\textbf{Surcharge} = P_{\text{acq}} - \textbf{EV}$) based on transaction velocity. The allocation is stored as an ETH-denominated allowance and is converted to $JUNK$ only when the purchaser claims it through the rewards module.

Let $\Delta t = t_{\text{curr}} - t_{\text{last\_acq}}$ be the time gap since the last acquisition pull.

$$\textbf{Purchaser Share Bps } \sigma(\Delta t) = \begin{cases} 0 & \textbf{if } \Delta t \leq \textbf{hotGap (60s)} \\ 10000 & \textbf{if } \Delta t \geq \textbf{coldGap (3600s)} \\ 10000 \cdot \frac{\Delta t - 60}{3600 - 60} & \textbf{otherwise} \end{cases}$$



- **High Activity ($\Delta t \leq 60$s)**: the purchaser allocation is 0% of the surcharge.
- **Low Activity ($\Delta t \geq 3600$s)**: the purchaser allocation is 100% of the surcharge.

Implemented Fee Slicing:

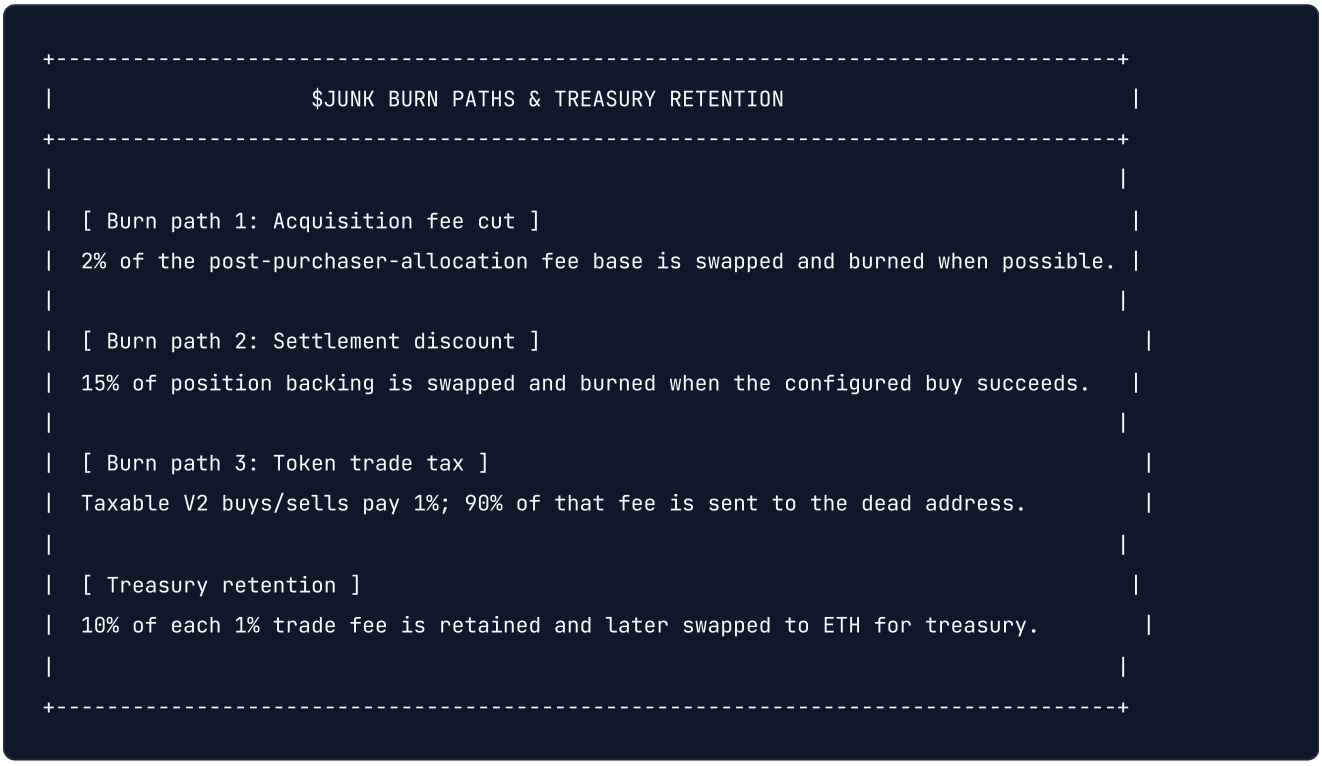
For an escrowed fee F, the contract computes the EV component $E = F \cdot 10000/(10000 + \textbf{surchargeBps})$ and purchaser slice $S = (F - E) \cdot \sigma(\Delta t)$. It then applies the following percentages to the **remaining amount** $F - S$, not only to the surcharge:

- **2% burn cut**: swapped to $JUNK$ and sent to the dead address when the swap succeeds.
- **1% team cut**: swapped to $JUNK$ and credited to the protocol payout balance.
- **Remaining dynamic cut**: credited as ETH pro-rata to active listings through `accFeePerEV`, or accrued to the protocol if there are no fee-share listings.

Consequently, the current implementation is not a simple 1%/2%/7% split of the surcharge alone; the exact amounts depend on the purchaser share and the remaining fee base.

4.4 $JUNK Burn Paths and Treasury Fee Retention

The current implementation has three protocol-generated dead-address burn paths, plus a retained treasury share of the token trade tax:



The trade-tax split is therefore equivalent to **0.90% of a taxable trade being burned** and **0.10% being retained for treasury**, subject to integer rounding. Fee-excluded addresses do not pay this tax. The public `burn(amount)` function also permits direct dead-address burns outside these three protocol-generated categories.

4.5 On-Chain Burn Accounting

`JunkWorldAssets.totalBurned()` returns the token balance held at `0x000000000000000000000000000000000000dEaD`. It includes acquisition burns, settlement-discount burns, the burned 90% share of token trade taxes, direct calls to `burn()`, and tokens transferred directly to the dead address. Because burning is implemented as a transfer to the dead address, the nominal ERC-20 `totalSupply` remains unchanged.

---

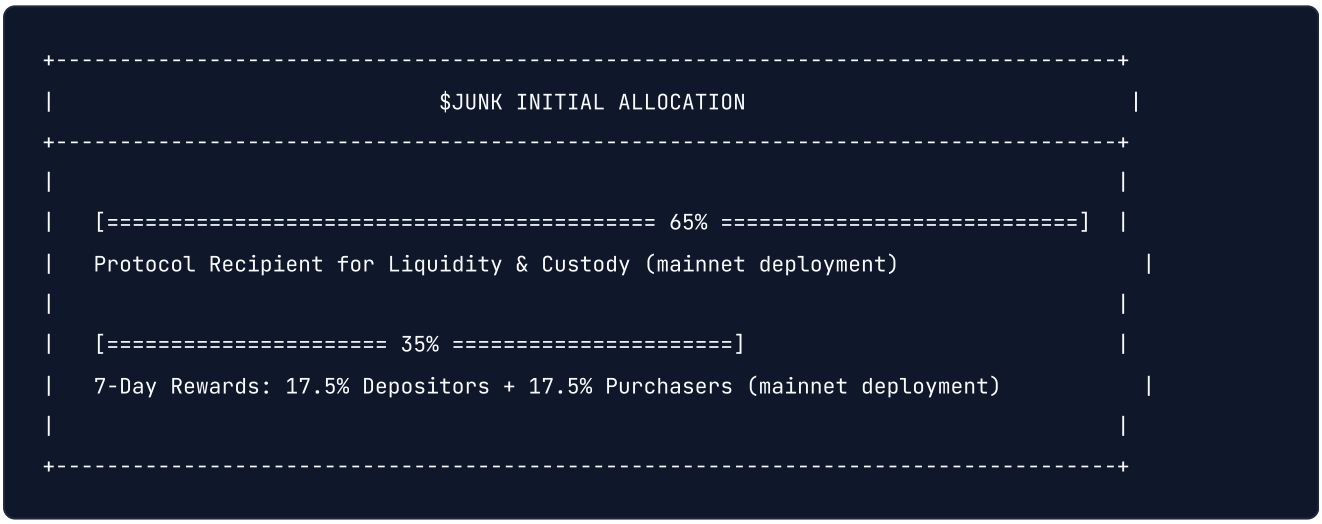
5. Tokenomics & Incentive Emissions

5.1 $JUNK ERC-20 Token Specification

- **Token Name**: Junk World Assets
- **Symbol**: `$JUNK`
- **Decimals**: 18
- **Total Initial Supply**: 1,000,000,000 `$JUNK`, minted once to the protocol address; no public mint function.
- **Dead Address**: `0x000000000000000000000000000000000000dEaD`

Supply Allocation:

- **Initial mint**: the token contract mints all 1,000,000,000 $JUNK to the protocol address.
- **Mainnet deployment allocation**: the deployment script transferred 350,000,000 $JUNK (35%) to `JUNKRewards`—175,000,000 for depositors and 175,000,000 for purchasers—and transferred the remaining 650,000,000 $JUNK (65%) to the protocol recipient for liquidity and custody. This split is a deployment action, not a permanent allocation rule enforced by `JunkWorldAssets.sol`.



5.2 7-Day Protocol Emissions Engine

`JUNKRewards.sol` executes a configured seven-day **fixed-rate** emission program once the JUNK owner enables acquisitions and the rewards module starts emission. The mainnet deployment is configured for 175,000,000 `$JUNK` in depositor emissions and 175,000,000 `$JUNK` in purchaser emissions. As of the deployment snapshot above, emission has not started.

$$\textbf{Emission Duration: } T_{\text{emission}} = 7 \text{ days} = 604,800 \text{ seconds}$$

Emissions are split into two parallel incentive pools:

1. Square-Root Backing Depositor Incentive Pool

Depositor emissions stream continuously based on the square root of position backing ($\sqrt{V_i}$):

$$\text{Rate per second: } R_{\text{dep}} = \frac{\text{DepositorTotal}}{604,800}$$

$$\text{Accrual Accumulator: accTokenPerSqrt} \leftarrow \text{accTokenPerSqrt} + \frac{R_{\text{dep}} \cdot \Delta t \cdot 10^{36}}{\sum \sqrt{V_i}}$$

$$\text{Depositor } i \text{ Pending Tokens} = \sqrt{V_i} \cdot \text{accTokenPerSqrt} - \text{tokenDebt}_i$$

The contract distributes the configured depositor rate pro-rata by $\sqrt{V_i}$; this is an allocation rule, not a guarantee against manipulation or economic loss. Rewards are claimable through the rewards contract rather than automatically transferred to a wallet.

2. Daily Purchaser Epoch Pot

Purchaser emissions are partitioned into 7 daily epochs ($\mathbf{1\ epoch = 86,400s}$):

$$\text{Daily Pot} = \frac{\text{PurchaserTotal}}{7}$$

At epoch close, purchasers claim tokens pro-rata based on their acquisition pull count:

$$\text{Purchaser Reward} = \text{DailyPot} \cdot \frac{N_{\text{user\_acquisitions}}}{N_{\text{total\_epoch\_acquisitions}}}$$

---

6. Security, Governance & Fail-Safe Architecture

6.1 Best-Effort Non-Blocking Asset Delivery

A primary vulnerability in traditional NFT escrow contracts is "griefing via revert" (e.g., a malicious recipient contract rejecting NFT transfers in `onERC721Received`).

`JUNK.sol` implements a **Best-Effort Delivery Strategy** (`_deliverAsset`):

```
try ERC721(collection).transferFrom(address(this), recipient, amountOrId) {
    // Delivery succeeded cleanly
} catch {
    stuckNFTRecipient[listingId] = recipient;
    emit NFTDeliveryFailed(listingId, recipient, collection, amountOrId);
}
```

If an asset transfer fails due to pausable collections, blacklisting, or fallback reverts:

1. The asset is recorded under `stuckNFTRecipient[listingId] = recipient`.
2. Core settlement state and ETH legs proceed; failed asset delivery is recorded for later recovery, while external swap failures are caught and may result in no tokens burned.
3. The recipient can withdraw their asset independently via `recoverStuckNFT(listingId)`.

6.2 Staging Queue & Activation

`JUNK.sol` features a staging queue, but the source does not encode a 24-hour pre-launch timer:

- Listings are staged when an unsettled acquisition exists or another staged listing is already queued; a listing can otherwise activate immediately.
- A staged listing can be withdrawn only while exits are unlocked and while it is not reserved for a sequence.
- Staged listings are activated by acquisition/processing flow in bounded batches; enabling `acquisitionsEnabled` starts emissions but does not itself drain the queue.

6.3 Emergency Rescue & Re-Entrancy Hardening

- **Re-Entrancy Protection**: Listing, acquisition, settlement, and withdrawal flows use Solady `ReentrancyGuard` where applicable (`nonReentrant`).
- **Checks-Effects-Interactions (CEI)**: Core settlement paths update listing and tree state before their external ETH or token transfers.
- **Pull-Payment Design**: Refunds and fee earnings are credited to internal mapping balances (`acquisitionRefundCredit`, `feeCredit`), eliminating block denial-of-service risks.

7. Smart Contract Interface Reference

`JUNK.sol` Core Interface

```
interface IJUNK {
    // --- Listing & Management ---
    function listNFT(address collection, uint256 tokenId) external payable returns (uint256 listingI
    function listTokenBag(address tokenAddress, uint256 tokenAmount) external payable returns (uint2
    function withdrawListing(uint256 listingId) external;
    function updateBacking(uint256 listingId, uint256 newBacking) external payable;

    // --- Acquisition ---
    function acquisitionFee() external view returns (uint256);
    function acquire(uint256 maxAcquisitionFee, uint256 minWeightedValue) external payable returns (
    function acquireWithSlippage(uint256 maxAcquisitionFee, uint256 minWeightedValue, uint256 maxNeg
    function acquireBatch(uint256 count, uint256 maxAcquisitionFee, uint256 minWeightedValue) exterr
    function processAcquisitions(uint256 maxCount) external returns (uint256 processedCount);

    // --- Settlement Choices ---
    function keepNFT(uint256 listingId) external;
    function acceptDepositorBid(uint256 listingId) external;
    function acceptBidAsTokens(uint256 listingId, uint256 minTokensOut) external;
    function relistNFT(uint256 listingId) external payable returns (uint256 newListingId);
    function depositorReclaimBacking(uint256 listingId) external;
    function depositorReclaimNFT(uint256 listingId) external;

    // --- Earnings & Fail-Safe ---
    function claimListingFees(uint256[] calldata listingIds) external returns (uint256 total);
    function withdrawEarnings() external returns (uint256 total);
    function withdrawAcquisitionRefund() external returns (uint256 amount);
    function recoverStuckNFT(uint256 listingId) external;
}
```

`JunkWorldAssets.sol` Core Interface

```
interface IJunkWorldAssets {
    function burn(uint256 amount) external;
    function totalBurned() external view returns (uint256);
    function StartTrading(address pair) external;
    function setTreasury(address newTreasury) external;
    function setSwapThresholdEth(uint256 newThreshold) external;
}
```

`JUNKRewards.sol` Core Interface

```
interface IJUNKRewards {
    function tokenShareBps(uint256 gap) external view returns (uint256);
    function claimAccruedTokens(uint256 minOut) external returns (uint256 tokenOut);
    function claimDepositorTokens(uint256[] calldata listingIds) external returns (uint256 total);
    function claimEpochTokens(uint256[] calldata epochs) external returns (uint256 total);
    function buyFor(address recipient, uint256 minOut) external payable returns (uint256 tokenOut);
}
```

---

8. Conclusion

The Junk World Assets (JUNK) Protocol provides a configurable framework for randomized position acquisition and yield distribution. It combines inverse-backing selection weights, an integer approximation of harmonic-mean pricing, Chainlink VRF V2.5 draw fulfillment, acquisition and settlement buy-and-burn paths, and a token trade tax whose fee is split 90% to the dead address and 10% to treasury retention. The resulting behavior depends on activation state, configuration, liquidity, and successful external swaps; these are not unconditional guarantees.

End of Technical Whitepaper.