Revnets

Filip
f@filip.world

September 22, 2023

Abstract

Smart contracts have enabled new models for organization and governance, but have in some ways failed to deliver on the promise of truly decentralized coordination. Among other causes, this is often due to the challenge of bootstrapping an organization while creating a product. The centralized nature of this process often leaves the founding team or core developers with little incentive to distribute power by the time a community has been formed. This leaves both parties worse-off: founders must contend with the day-to-day overhead of running a DAO or a governance system which has little to do with their end product, and participants must contend with the risk of misalignment, rugpulls, and scams. **Revnets** are a new model for creating, scaling, and operating decentralized networks according to pre-defined economic incentives which align participants and remove the need for governance, freeing founders from day-to-day management and making rugpulls impossible.

1 Mechanism

Each Revnet accepts a specific currency, chosen when the network is created. Anyone can pay a Revnet with its currency to issue the Revnet's tokens, or they can destroy tokens to reclaim currency from the Revnet. For illustrative purposes, this section describes a network which accepts ETH and issues a $token.[1]

Under the Revnet's initial conditions, payers receive 1 $token per 1 ETH. $token issuance evolves over generations which last a pre-defined length of time (1 day, for example). Three mechanisms determine how $tokens behave:

- The **Price Ceiling** is the ETH price to create new $tokens, expanding the $token supply. The price ceiling increases at a fixed rate each generation, making network expansion more expensive over time.

[1] In practice, Revnets can accept and denominate prices in other currencies. For a clearer understanding of USD-based accounting, see Section 4.2.

- The **Price Floor** is the ETH value that can be reclaimed from the network by destroying $tokens, contracting the $token supply. The price floor increases dynamically as the network contracts or expands.

- The **Price Window**. Liquidity can be added to a $token↔ETH market[2] at any time. While the $token's market price is between the price ceiling and price floor, $tokens are purchased and sold on the market. Otherwise, purchases and sales are fulfilled at the price ceiling or price floor, expanding or contracting the $token supply in response to demand.

In practice, Revnets issue $tokens at the price ceiling until a $token↔ETH market forms. From that point onwards, $tokens are purchased and sold on the market until the price reaches the price ceiling or price floor, at which point the $token supply will expand or contract to meet market demand.

Revnets can also specify a **Boost** which routes a percentage of purchased $tokens to a specific address for a pre-determined length of time after the network's creation. This could be a developer multisig, a staking rewards contract, an airdrop stockpile, or something else. Network creators also have the option to pre-mint an arbitrary number of $tokens to this address when the network is created.

Once a Revnet is deployed, these parameters are locked in place.

1.1 Price Ceiling

The cost to expand the $token supply increases over time. This is done by reducing the number of $tokens issued per ETH with each passing generation. This reduction is dictated by a price ceiling function, which compounds at a rate r_c set by the network's creator.

The price ceiling function[3] can be expressed as:

$$T_n = T_1 \cdot (1 - r_c)^{(n-1)} \tag{1}$$

where:

- T_n is the number of tokens issued per 1 ETH in the n^{th} generation,

- T_1 is the number of tokens issued per 1 ETH in the first generation,

- r_c is the price ceiling curve, or rate of decrease per generation (expressed as a decimal), and

- n is the generation number, which increments sequentially starting from 1.

Since Revnets have a T_1 (initial price) of 1 $token per 1 ETH, this can be simplified to:

[2] Our initial implementation uses a Uniswap v3 liquidity pool.
[3] Also see the interactive price ceiling function on Desmos

$$T_n = (1 - r_c)^{(n-1)} \tag{2}$$

The price ceiling encourages participants to join the network early. The earlier a participant joins, the more $tokens they receive for their ETH.

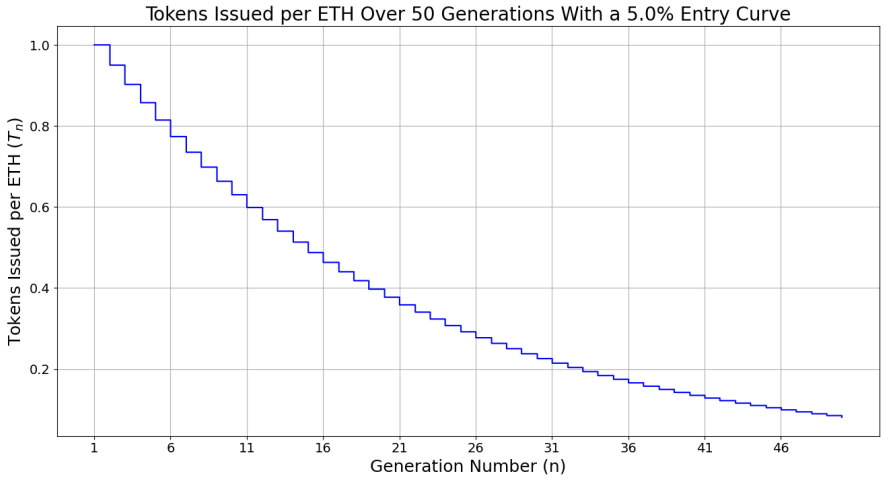

Figure 1: This figure shows how T_n (the number of $tokens issued per 1 ETH in the n^{th} generation) varies across 50 generations with a 5% price ceiling curve ($r_c = 0.05$). Note that T_n decreases rapidly at first, then more gradually as T_n tends towards 0 over many generations.

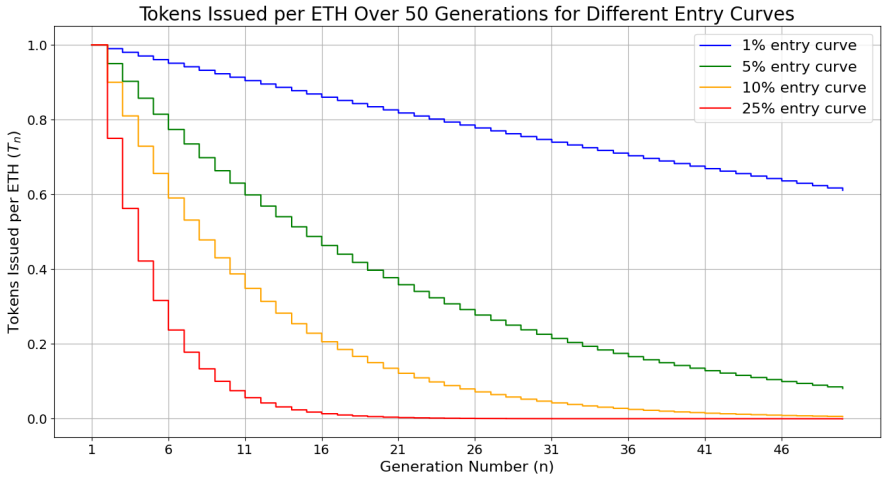

Figure 2: This figure shows how T_n varies over 50 generations with 1%, 5%, 10%, and 25% price ceiling curves ($r_c = 0.01, 0.05, 0.1, 0.25$). Note that for greater entry curves, T_n tends towards 0 more quickly.

1.2 Price Floor

Anyone can destroy their $tokens to reclaim ETH from the network. The amount of ETH they can reclaim is determined by the price floor function, which is dynamically calculated based on a price floor curve r_f set by the network's creator, the total $token supply, and the amount of ETH in the network.

The price floor function[4] can be expressed as:

$$V_r = \frac{V_t \cdot x}{s}\left((1 - r_f) + \frac{r_f \cdot x}{s}\right) \tag{3}$$

where:

- V_r is the amount of ETH which gets reclaimed,
- V_t is the total amount of ETH in the network,
- s is the total $token supply,
- r_f is the price floor curve (expressed as a decimal), and
- x is the number of $tokens being burned.

Unless r_f is 0, this function ensures that as more $tokens are destroyed (i.e., the larger x is), the more ETH is reclaimed per token.[5] This means the first participants to destroy their $tokens get less ETH per $token than participants

[4] Also see the interactive price floor function on Desmos

[5] If r_f is 0, each $token will yield the same amount of ETH when destroyed.

who destroy their $tokens later. This incentivizes participants to stay in the network longer than other participants in order to increase the amount of ETH they can reclaim.

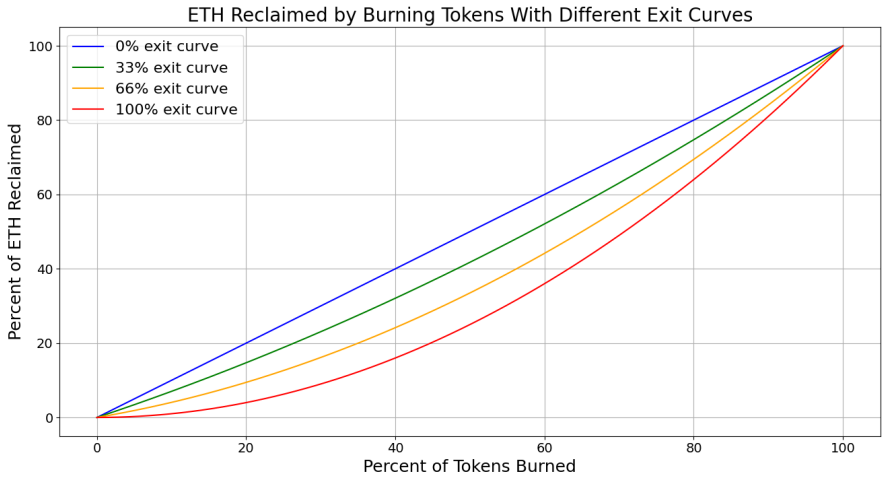

Figure 3: This figure shows price floor functions with price floor curves of 0%, 33%, 66%, and 100% ($r_f = 0, 0.33, 0.66, 1$). The x axis represents the percentage of the total $token supply being destroyed, and the y axis represents the percentage of the network's ETH which is reclaimed. Note that with a 0% price floor curve ($r_f = 0$), reclaim amounts are linear with respect to the number of tokens being destroyed—a network participant which destroys *e.g.* 10% of the total token supply can reclaim 10% of the network's ETH. The larger the price floor curve, the less ETH earlier reclaimers receive.

2 Contract Implementation

Revnets are implemented as Solidity smart contracts, and are intended for use on Ethereum, Ethereum L2s, and other environments with Solidity support. Revnets are built on top of and extend the Juicebox[6] protocol. Revnets leverage Juicebox's discount rate logic for price ceiling calculations, its redemption logic for price floor calculations, and its reserved rate logic for boost calculations.

While the price of a Revnet's token is between the price ceiling and the price floor, payments are routed to a Uniswap v3 liquidity pool by a Buyback Delegate contract if possible.[7] The Buyback Delegate contract routes incoming payments to the Uniswap pool if doing so would yield more tokens than a payment into the network would create.

[6] See the Juicebox Docs.
[7] See the Buyback Delegate contracts on GitHub.

Each Revnet is a Juicebox project, which is deployed, owned, and administrated by one of the contracts below.

Contract Name	Description
`BasicRetailistJBDeployer`	Deploys a basic Revnet.
`PayAllocatorRetailistJBDeployer`	
`Tiered721PayAllocatorRetailistJBDeployer`	

Table 1: The Revnet deployer contracts.

2.1 BasicRetailistJBDeployer

Anyone can deploy a basic Revnet by calling the `BasicRetailistJBDeployer` contract's `deployBasicNetworkFor(...)` function:

```
function deployBasicNetworkFor(
  address _operator,
  JBProjectMetadata memory _networkMetadata,
  string memory _name,
  string memory _symbol,
  BasicRetailistJBParams memory _data,
  IJBPaymentTerminal[] memory _terminals,
  BuybackDelegateSetup memory _buybackDelegateSetup
)
  public
  returns (uint256 networkId)
{ ... }
```

This function accepts the following parameters:

- `_operator` An address that will receive the boost and any pre-minted token. This address has permission to change the boost recipient(s).
- `_networkMetadata` The Revnet's metadata content and domain (for use in clients).
- `_name` The name of the ERC-20 token being created for the Revnet.
- `_symbol` The symbol of the ERC-20 token being created for the Revnet.
- `_data` A `BasicRetailistJBParams` struct which specifies the initial token issuance rate for the first generation, the number of tokens which should be pre-minted to the boost address, the generation length, the entry and exit curves, and the boost percentage and duration.
- `_terminals` The Juicebox payment terminals that the network uses to accept payments.
- `_buybackDelegateSetup` Info for setting up the buyback delegate to use when determining the best price for new participants.

This function deploys and configures the Revnet, deploys the Revnet's ERC-20, premints tokens as needed, and grants the appropriate permissions to the boost address. It then returns a `networkId`, the Juicebox project ID of the newly created Revnet.

3 Applications

3.1

3.2 Software

Revnets

3.3 Social Networks

4 Miscellanea

4.1 Network Token

4.2 Accounting Types

If the network uses USD-based accounting, payers will receive 1 token per USD worth of ETH under the network's intial conditions. For instance, if the ETH price is 1,000 USD, paying 1 ETH into the network will yield 1,000 tokens.

4.3 L2s