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 has its own token (the *network token*) which represents a claim on the network's revenue. Each Revnet accepts a currency, which people can pay the network with to buy its token (thus joining the network). For illustrative purposes, this section describes a network which accepts ETH and issues a *$token*.[1]

Revnets have no owner. Once they are deployed, their parameters are locked in place. Funds can only leave the network when people exit.

Under the network's initial conditions, payers receive 1 $token per ETH. A Revnet's $token issuance evolves over generations which last a pre-defined length of time (28 days, for example). Three mechanisms determine how $tokens can be purchased and sold:

- The **Price Ceiling**. The price to create new $tokens, thus 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**. The amount of ETH that can be reclaimed from the network by destroying $tokens, thus contracting the $token supply. The price floor increases dynamically as the network contracts or expands over time.
- The **Price Window**. Liquidity can be added to a $token↔ETH automated market maker (AMM) pool at any time. While the $token's price is between the price ceiling and price floor, purchases and sales are fulfilled by the pool if possible. Otherwise, purchases and sales are fulfilled at the price ceiling and price floor.

Revnets can also specifiy 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.

1.1 Price Ceiling

The cost of acquiring a network's tokens becomes more expensive 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_{en} set by the network's creator.

The entry curve function[2] can be expressed as:

$$T_n = T_1 \times (1 - r_{\text{en}})^{(n-1)} \tag{1}$$

where:

- T_n is the number of tokens issued per ETH in the n^{th} generation,
- T_1 is the number of tokens issued per ETH in the first generation,
- r_{en} is the *entry 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 always have a T_1 (initial price) of 1 token per ETH, this can be simplified to:

$$T_n = (1 - r_{\text{en}})^{(n-1)} \tag{2}$$

The entry curve mechanism provides an incentive for participants to join the network early. The earlier a participant joins, the more tokens they receive for their ETH.

[2] Also see the interactive entry curve function on Desmos

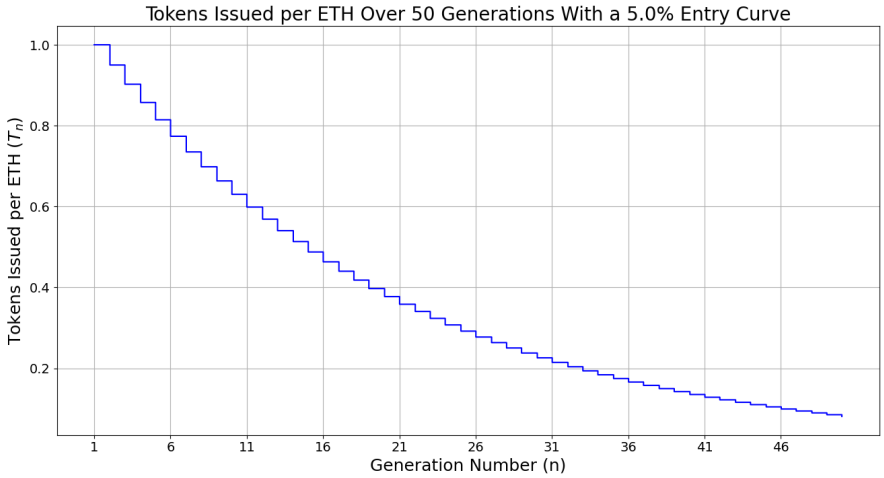

Figure 1: This figure shows how T_n (the number of tokens issued per ETH in the n^{th} generation) varies across 50 generations with a 5% entry curve ($r_{\text{en}} = 0.05$). Note that T_n decreases rapidly at first, then more gradually as T_n tends towards 0 over many generations.

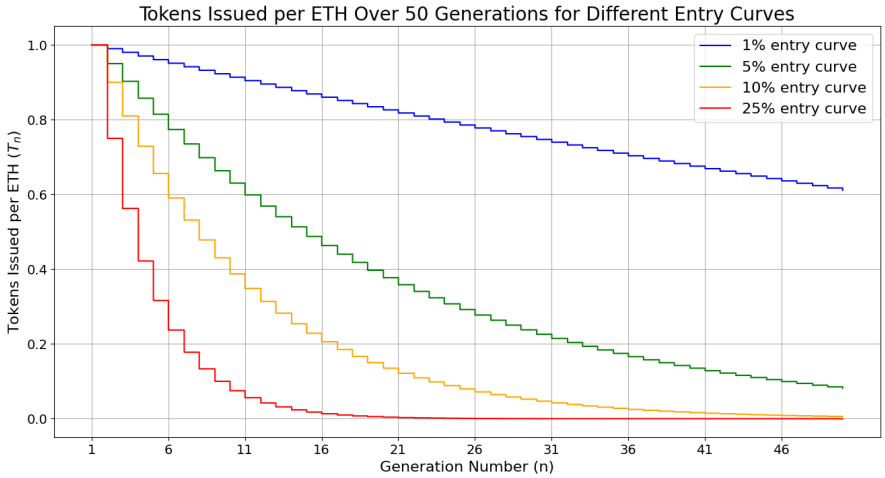

Figure 2: This figure shows how T_n varies over 50 generations with 1%, 5%, 10%, and 25% entry curves ($r_{\text{en}} = 0.01, 0.05, 0.1, 0.25$). Note that for greater entry curves, T_n tends towards 0 more quickly.

1.2 Price Floor

Anyone can exit the network by burning their tokens, which allows them to reclaim some of the network's ETH. The amount of ETH which can be reclaimed is determined by the exit curve r_{ex}, which is set by the network's creator.

The exit curve function[3] can be expressed as:

$$V_r = \frac{V_t \times x}{s}\left((1 - r_{\text{ex}}) + \frac{r_{\text{ex}} \times x}{s}\right) \tag{3}$$

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 supply of tokens,
- r_{ex} is the *exit curve*, and
- x is the number of tokens being burned.

This function ensures that the more tokens are burned (i.e., the larger x is), the more ETH gets reclaimed per token. This means the first participants to exit a network get less ETH per token than participants who exit later. This incentivizes participants to stay in the network longer than other participants in order to increase the amount of ETH they can reclaim by exiting.

[3] Also see the interactive exit curve function on Desmos

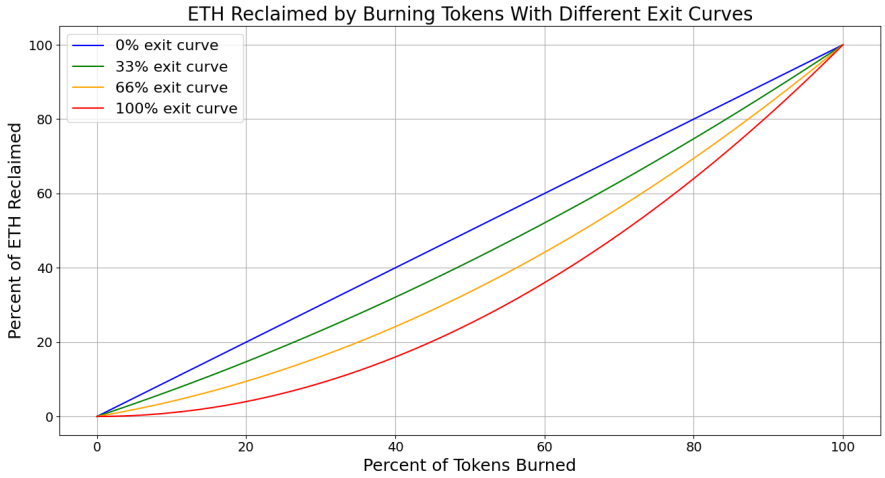

Figure 3: This figure shows exit curve functions for exit curves of 0%, 33%, 66%, and 100% ($r_{\text{ex}} = 0, 0.33, 0.66, 1$). The x axis represents the percentage of the total token supply being burned, and the y axis represents the percentage of the network's ETH which is reclaimed. Note that with a 0% exit curve ($r_{\text{ex}} = 0$), reclaim amounts are linear with respect to the number of tokens being burned—a network participant which exited by burning *e.g.* 10% of the total token supply would reclaim 10% of the network's ETH. The larger the exit curve, the less ETH earlier exiters receive.

1.3 Boost

For a pre-determined length of time (140 days, for example) after a Revnet's creation, a percentage of newly generated tokens are allocated to a specific address. This address could be a developer multisig, a staking rewards contract, an airdrop stockpile, or something else. When the boost period ends, this allocation drops to 0.

1.4 Buyback

Revnets also have the option to specify a *Buyback Delegate*[4] contract. A Buyback Delegate contract maximizes the number of tokens that a payer will receive by routing incoming payments to a Uniswap liquidity pool if doing so would yield more tokens than a payment into the network.

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.

[4]See the Buyback Delegate contracts on GitHub.

Revnets are built on top of and extend the Juicebox[5] protocol, thus inheriting the risks and security of the Juicebox protocol. Revnets leverage Juicebox's discount rate logic for entry curve calculations, its redemption logic for exit curve calculations, and its reserved rate logic for boost calculations.

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.

[5] See the Juicebox Docs.

- `_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

5 Conclusion