Døllar

Empty Set Squad
emptysetsquad@protonmail.com

August 24, 2020

Abstract

Most realizations of decentralized stablecoins so far have been based on collateral reserve models. These work reasonably well, but are both capital inefficient and carry risk to the underlying collateralized assets. Additionally, the total supplies of these stablecoins are constrained to strictly less than the available reserve assets on-chain. In this paper we'll propose instead an elastic supply stablecoin constructed on existing primitives, and describe its implementation as a fully decentralized Ethereum DAO.

1 Prior Work

Instead of reinventing the wheel, we aim to construct Døllar using already existing primitives where possible. This allows for a minimal implementation with few unknown unknowns.

Døllar's stability mechanism is an iteration on **Basis** [1], while borrowing the notion of epochs from **0x Staking** [2]. We use a special AMM to price coupon premiums inspired by **Rho** [3], and construct Døllar's initial price oracle directly on **Uniswap v2** [4].

2 Contract Architecture

The Døllar protocol is operated by a DAO that governs and regulates the supply of its stablecoin `ESD`. Its DAO utilizes a price oracle contract built ontop of the `ESD:USDC` Uniswap v2 pool. This modular design allows for easy upgrades as Døllar's ecosystem becomes more robust.

3 Epoch

Døllar's DAO splits time into distinct epochs of roughly one day to simplify logic around governance, supply regulation, and flash loan resistance.

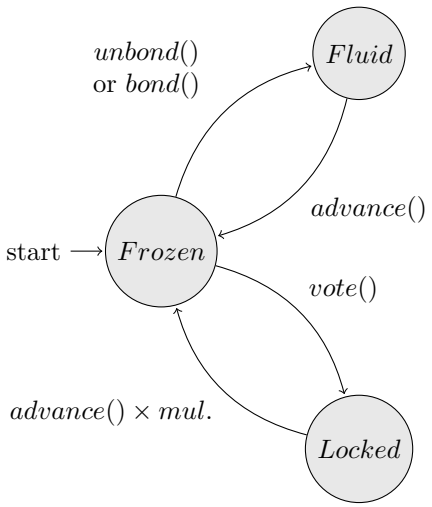
Epochs are *advanced* manually by sending an `advance()` transaction to the DAO. This allows us to apply state transformations like coupon expiry and supply regulation on advancement. To incentivize this behavior, the DAO mints reward `ESD` tokens to the sender upon successful advancement.

Epochs are available for advancement based on the current block timestamp, however there are no guarantees that an epoch will be advanced immediately when available.

4 Bonding

Døllar is a single token protocol. In addition to being the stable asset, `ESD` tokens are also used to gain ownership in its DAO.

Users deposit `ESD` tokens into the DAO which can then be bonded. Bonded tokens grant the owner stake in the DAO, but are temporarily locked from depositing or withdrawing until at least the following epoch. This ensures a static balance during the bonded epoch which underpins its security model.

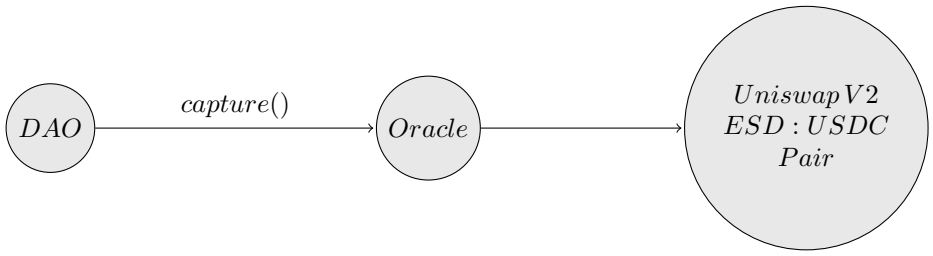


We introduce a special state `Locked` to cover cases where our DAO must lock already-bonded user funds for multiple epochs. A user enters this state when voting during the governance process, and remains there until the election has concluded to ensure the user's balance is unchanged for the duration.

To summarize, the user can only bond and unbond when their account is `Frozen` or `Fluid`, but can deposit, withdraw, vote, and receive regulator payouts when they are `Frozen` or `Locked`.

5 Oracle

Døllar's initial oracle is constructed using the `ESD:USDC` Uniswap v2 pair, giving us a flash loan resistant price feed out of the box. New average weighted prices are polled and computed from the oracle's data during each epoch advancement cycle.



5.1 Incentivization

Oracle liquidity providers incur an opportunity cost by keeping their `ESD` outside of the DAO. To offset this, a small portion of the newly minted `ESD` during a supply extension will be diverted to an LP incentivization pool. Rewards in this pool will be distributed to liquidity providers who have staked their `UNI-V2` tokens from the `ESD:USDC` pair.

6 Stability Mechanism

Døllar uses supply elasticity to adjust for changing demand while targeting a fixed value per token. We balance supply with the following simple equality,

$$supply_n \cdot price_n = supply_{n+1} \cdot 1.00$$

From which we can determine the required change in supply to regulate the price,

$$\Delta supply = supply_n \cdot (price_n - 1.00)$$

Additionally, to prevent *debt* from overtaking *supply*, we instead use *net supply* (*supply* less *debt*) in the above calculation which allows *debt* to approach *supply* asymptotically.

6.1 Supply Extension

For positive $\Delta supply$, new `ESD` tokens are minted by the DAO and distributed as follows:

1. Credited to redeemable pool, for outstanding coupons, until pool can cover entirety of the outstanding coupons.

2. Immediately burned to reduced *debt* until *debt* reaches 0.
3. Any surplus credited pro rata to bonded `ESD` holders.

6.2 Supply Contraction

In order to contract the supply, we must incentivize participants to burn their `ESD` tokens. This is accomplished by offering coupons redeemable for a premium quantity of future `ESD`.

To contract by $\Delta supply$, the DAO issues the equivalent amount of *debt*, which marks the current amount of `ESD` tokens that may be burnt in exchange for `coupons`.

7 Coupon Market

To incentivize the exchange of `ESD` tokens for coupons, we create an AMM over the current *debt ratio*. As the debt ratio increases, so does the coupon premium, which ensures proper incentivization even during periods of extreme down-regulation.

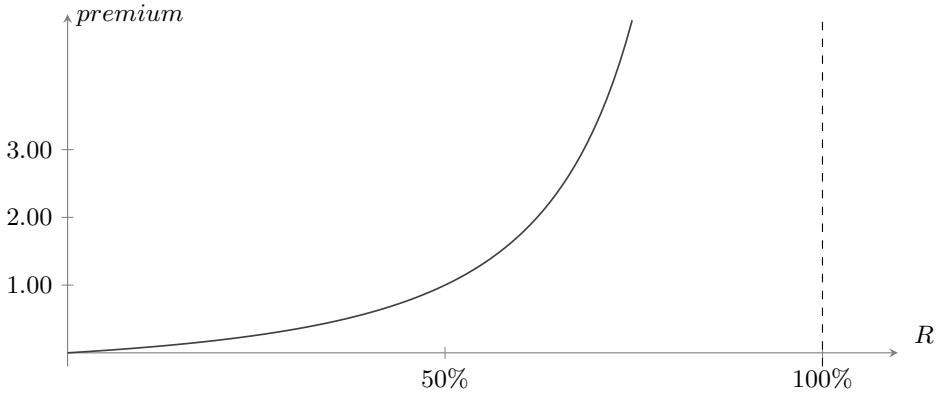
It's important to note that the debt-ratio is *expected* to be non-zero during periods of supply contraction as it is the mechanism for which the market prices the coupon premium.

First we define the *debt ratio* R as,

$$R = \frac{debt}{supply}$$

An ideal premium curve would have *zero* premium at *zero* debt, with asymptotically increasing premium as debt approaches supply. With this in mind, we can define our premium curve over R as,

$$premium(R) = \frac{1}{3(1-R)^2} - \frac{1}{3}$$



7.1 Calculating coupon amount

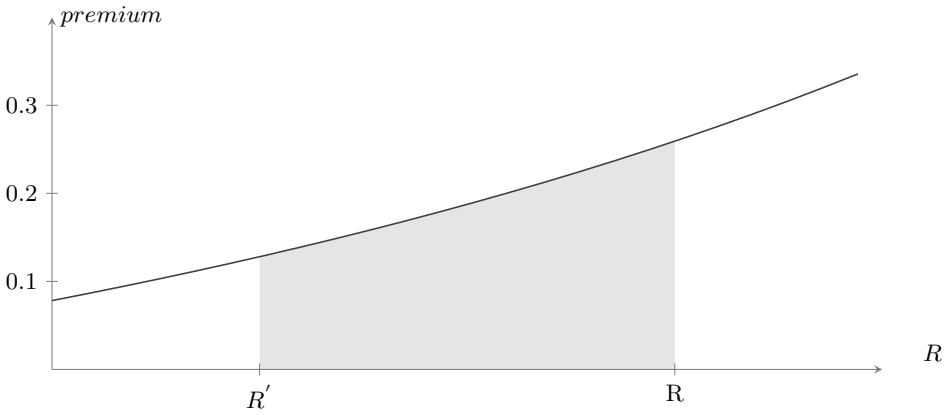
With our curve defined, we can examine how an order of size n will be priced. We begin by defining $R^{'}$, the resulting debt ratio after execution as,

$$R^{'} = \frac{debt - n}{supply - n}$$

With $R^{'}$, we can find our order premium by taking the mean over the curve from R to $R^{'}$.

$$premium = \frac{\int_{R'}^{R} premium(R)\, dR}{R - R'} \tag{1}$$

$$= \frac{1}{3(1-R)(1-R')} - \frac{1}{3} \tag{2}$$



Once the premium is determined we can calculate the resulting coupon amount for the order.

$$coupons = n \cdot (1 + premium) \tag{3}$$

7.2 Redemption

Coupons entitle holders to a 1:1 redemption of `ESD` tokens at some point in the future. When there is a supply growth event, new `ESD` tokens are first credited to the redeemable pool. This pool is first-come first-served for redemption by any current coupon holder.

This ensures that a queue does not form, disincentivizing new coupon purchasers, as debt increases. Further, coupons are valid for 90 days, after which they will expire and no longer be redeemable. These are key to mitigating downward spiral events prevalent in similar stability mechanism designs.

8 Governance

The DAO is fully self-governing at launch. A new DAO implementation may be proposed at any time by any current stakeholder with at least `1%` of the current DAO stake.

Voters have 7 days to choose to either `approve` or `reject` the candidate implementation. If a quorum of 33% is reached and more votes `approve`, then the new implementation may be committed.

By allowing only full implementation upgrades, even for simple constant modifications, we ensure a lightweight governance process.

9 Bootstrapping Mechanism

Three processes work together to bootstrap the protocol and distribute the initial batch of `ESD` tokens fairly to early protocol participants, which can then in turn be used to gain ownership over its DAO.

1. At each successful epoch advancement there is an incentivization payout of `100 ESD` tokens to the transaction sender.
2. For the first `30` days the price oracle will be fixed at `1.10 USDC` causing a `10%` supply increase each epoch, after which the oracle will function as normal, reporting the true market price.
3. During the bootstrapping phase, epochs will be shortened to `8 hours` to improve distribution as well as shorten governance turnaround as a precaution.

These work together to distribute a solid initial supply of `ESD` after which the protocol may either expand or contract going forward given demand.

References

[1] N. Al-Naji, J. Chen, L. Diao, Basis: A Price-Stable Cryptocurrency with an Algorithmic Central Bank, (2018). https://www.basis.io/basis_whitepaper_en.pdf.

[2] 0x, 0x Staking Specification (v3), (n.d.). https://0x.org/docs/guides/v3-staking-specification.

[3] M. Wolff, Rho: Automated Interest Rate Swaps, (n.d.). http://maxcwolff.com/rho.pdf.

[4] H. Adams, N. Zinsmeister, D. Robinson, Uniswap v2 Core, (2020). https://uniswap.org/whitepaper.pdf.