Solid Protocol

0x2B6C7F44DD5A627496A92FDB12080162e368aB1E

0x734241200496E2962b1e2553e5b4FeB99347E1d0

soliddefi@gmail.com

Abstract— **This paper explains a new mechanism based on a new supply-demand model which allows exchanges between tokens and a stable version of the token itself. It also covers the technical details how such mechanism works, by using a price formula that allows minting and burning of token without an authority. Such a new high transparency supply-demand model is possible combined with the application of blockchain.**

Keyword— supply, inflation, blockchain, decentralize, algorithm

I. INTRODUCTION

We proposed a new mechanism, that supply is increased at the moment the market demands into it, decreased vice versa. Commonly, we think that price decreases as supply increases in the market, but if we implement a **"mint-on-buy" and "burn-on-sell" mechanism**, which is discussed later in this paper, the price could still increase as a norm when supply is added, because the supply only increases when there is demand, it balances out the supply accordingly to the demand.

Price fluctuate sometimes in all market affected by liquidity, as token's supply could not be balanced out in time [1]. Nearly all supply in markets we see nowadays are adjusted by humans, when it is needed, authorities would change the amount of supply by minting and burning supply, on a specific targeted time.

However, those changes that authorities made on the supply may not result as expected, due to supply and demand is changing in a dynamic way, and it is nearly impossible to change supply constantly from time to time without the application of an algorithm. But by combining the use of smart contract with an algorithm on the blockchain network, it is possible to create a decentralized market with supply constantly changing.

II. MECHANISM

The **Solid Protocol** acts as a bridge that could take any supply of any market and convert it into a **"Solid version"** of the market. It creates a stable market that the price is adjusted non-linearly, in which it is a more stabilized market and could be considered as a subset of the original market.

There are two key points in this **mechanism**:

- The **only way** Solid-token is minted and burnt is through buying and selling. (meaning there should be 0 supply at the beginning)
- Tokens are always minted when bought and burnt when sold **if and only if** through the **Solid Protocol**.

A. Solid protocol

When buyers buy into the Solid Protocol with a token on the blockchain, the "Solid version" of the coin is minted and sent to the buyer, total supply also increased by the amount that is minted.

Vice versa, when sellers sell the "Solid version" of a token on the blockchain, the original token is sent to the seller, the "Solid version" of the token is burnt, total supply also decreased by the amount that is burnt.

The amount that is minted/burnt always based on the supply at the moment when Solid-token is minted/burnt. Basically, the price is based on supply.

1) Self-liquidated

The **Solid Protocol** bridges all pairs of tokens into a "Solid version" of those tokens themselves. Every token could be converted into a **"Solid version"** of the token itself.

When the supply of a token is being minted, those supplies could then be used to buy into the **"Solid version"** of the token, creating a self-liquidated **Solid-token** based on the token itself.

By **"proof-of-staking**", holders of the **Solid-token** can lock up their liquidity in a pool to ensure there is a supply that would not be sold, and be rewarded as others sell. This creates a good cycle of a supply and demand model that there is a value of just holding the Solid-token itself.

Unlike most markets, the Solid Protocol does not adjust the price of the Solid-token linearly, but with a simple formula to calculate the price of such. (See below section 3 Price

Oracle)

2) Anti-inflation

Solid protocol allows traders to use **Solid-token** as a base for trading, it is also an anti-inflation protocol for the market with **quantitative easing (QE)**, since if one holds a **"Solid version"** of a stable coin, the **Solid-token** could keep itself value accordingly to inflation as the supply of the **"Solid version"** of the stable coin would increases when more stable coin is minted.

If one holds a stable coin like US dollars (USD), USD is being minted but the holder does not benefit from this as stable coin does not increases in value as time goes by generally, but by converting USD into Solid-USD through the Solid Protocol, the buyer holds the **"Solid version"** of the token which enables the catch up to inflation as more USD is being minted, the price of Solid-USD also increases over USD as more USD is being used to buy into Solid-USD.

However, a token that has a fixed supply that does not increase over time could not take good use of the **Solid Protocol** since there would be also a fixed amount in this protocol based on the fixed supply.

3) Price oracle

We propose a simple mechanism to calculate the price, based on supply, given x as the supply of the token, the price of the token, $f(x)$, is determined by the following formula:

$$f(x) = \sqrt[2]{x}$$

Formula 1.1 Price of Solid-token based on total supply of Solid-token

In a real-life scenario, when a trader buys the token, the trader cannot buy all the tokens he wants at the current price. Exchanges mimic the effect that, the token's price goes up as the buyer buys in more coin in a dynamic way, which is called **price impact**. Price impact normally could be deduced to a simpler formula since markets normally adjust the price linearly.

All price impact could be calculated with definite integral given $f(x)$ as the formula for calculating the price. Since the proposed method does not adjust the price in a linear way, therefore, we need to deduce our own price impact formula with the definite integral.

Let α be the initial supply of the Solid-token, β be the final supply of the Solid-token, and let δ = β - α, which is tokens in total bought by the buyer, the below formula is used to calculate the total price of all tokens that the buyer needs to pay for buying δ token(s). Given $f(x) = \sqrt[2]{x}$, we could further deduce the total price with the Riemann Sum definite integral formula:

$$\lim_{\Delta x \to 0} \sum f(x_i)\Delta x$$

Riemann Sum formula

$$\lim_{n \to \infty} \sum_{i=1}^{n} \frac{\delta}{n} * \sqrt[2]{\alpha + i * \frac{\delta}{n}}$$

or

$$\lim_{n \to \infty} \sum_{i=1}^{n} \frac{\delta}{n} * \sqrt[2]{\beta - i * \frac{\delta}{n}}$$

Formula 1.2. Definite integral summation formula of formula 1.1

$$\int_{\alpha}^{\beta} \sqrt[2]{x}\,dx$$

$$= \left[\frac{2}{3} x^{3/2}\right]_{\alpha}^{\beta}$$

Formula 2. Token based on δ amount of Solid-Token

For instance, let's call a token ABC, if one sells 5 Solid-ABC when the supply is 105 Solid-ABC, the supply of Solid-ABC becomes 100 after selling, the amount one would get is $\frac{2}{3} * (105^{\frac{3}{2}} - 5^{\frac{3}{2}}) \approx 709.83$ ABC (rounded to 2 decimal places), the same price as for buying 5 Solid-ABC when the supply is 105 Solid-ABC.

The below formula could be deduced for the total amount of token one can get, in a buying scenario, given p as the current supply and n as the amount in price, we would not provide the mathematical proof or discuss in detail how the below formula is deduced. The total number of tokens is deduced as follows:

$$(\frac{3}{2} n + p^{3/2})^{2/3} - p$$

Formula 3. A formula deduced for buying

4) Anti-dumping dynamic trading fee

The **Solid Protocol** also has embedded a feature that is against the attack of dumping and dumping of the price, it introduces a **dynamic trading fee** (similar to gas fee on the Ethereum blockchain network) based on the exponential daily trading volume.

Holders of the “Solid version” of a token could stake into the staking pool that collects those trading fees, when selling volume increases, the trading fee would also increase as holders of in the protocol is locking up their liquidity, it could be so-called **“proof by staking”**, arbitrageurs that has large proportion in the Solid Protocol could not profit from an instant-massive sell-off as the trading fee is increased.

On every single transaction of selling, the **exponential average trading volume** (EATV) would be updated based on the number of tokens sold and the difference between last traded timestamp t_1 and timestamp t_2 of the transaction when the transaction is executed, with the below formula:

Let δ be the difference between the last traded timestamp and timestamp of the transaction $(t_2 - t_1)$, and ρ be the number of tokens sold:

$$\mathrm{EAVT}_{new} = \begin{cases} \mathrm{EAVT}_{old} * (1800 - \delta) + \rho * \dfrac{\delta}{1800}, \delta < 1800 \ seconds \\ \rho, \delta \geq 1800 \ seconds \end{cases}$$

Formula 4. Calculation of EAVT based on δ

Let α be the balance of tokens in the Solid Protocol, then **the trading fee percentage (TFP)** is calculated based on the following:

$$\mathrm{TFP} = \min(\mathrm{EAVT} * 48 / \alpha), 1) * 20\%$$

Formula 5. Calculation of TFP based on α and EAVT

Finally, the **trading fee (TF)** sent to the reward pool is calculated as follows:

$$\mathrm{TF} = \mathrm{TFP} * \rho$$

Formula 6. Calculation of TF based on TFP and ρ

TF would be the amount sent to the reward pool and distributed among those who staked in the staking pool proportionally.

III. IMPLEMENTATION

Normally, on the Ethereum blockchain network, **ERC20** is most commonly used as a standard, creating a mapping of each ERC20 to the smart contract itself is not a standard in the Ethereum blockchain network, therefore we have created a new standard called **ERC20-multi.**

Since each token has its own contract address, by using ERC20-multi, we could hold the information of each token.

Each token has the following attribute mapped in the **Solid protocol**:

- Amount of the “Solid version” of the token (supply of Solid)
- Exponential average trading volume (EATV)
- Last traded timestamp

Individuals can buy with any ERC-20 token in the Ethereum blockchain network, the amount that one can get is calculated based on the mechanism stated in the above session of this paper.

A. Functions

Mathematical calculation is expensive on the Ethereum blockchain network, which could consume a lot of gas. Calculation such as square root and cubic root could be costly. Therefore, the most optimized algorithm of such functions should be used.

The discussion below is based on Solidity in the Ethereum blockchain network.

1) Square root function

The square root function used here (a fast square root algorithm by bit shift) is different from the one used in the UNISWAPV2 contract in the Ethereum blockchain network (Babylonian method), which is generally consuming less gas.

For big number such as 1000000000000000, the Babylonian method consumes 3569 gases, while the square root function here only consumes 1008 gases. The Babylonian method generally consumes more gas but only consumes slightly less when the input is small (< 273) compared to the square root function.

2) Cubic root function

The cubic root function adapted an algorithm (with quadratic convergence) [2], which is generally consuming less gas than methods such as the nth-Newton-Raphson method and binary search.

B. Initialization

As there could be a **division-by-0 error** when the supply is 0, there are minimum values staked balance and supply that should be set to avoid such error.

The default value for pointer mapping storing data is always 0 in solidity and could not be changed, therefore, a function is created so that this function must be called to assign the minimum values accordingly before any trade takes place in all ERC20 token.

IV. PRECISION

Since doing mathematical computation such as cubic root, allocating extra bits for computation to avoid overflow could be costly, a specific precision is chosen here (divided by 10**14) for the calculation of selling before the cubic root function is called.

Due to the asymmetry feature of the mathematical implementation in the mintOnBuy() and burnOnSell() functions, the precision is different in the two functions.

A. Overflow

Although in the Ethereum blockchain network most ERC20 tokens use uint256 to store the supply and the balance of individuals, the Solid Protocol won't be able to max at uint256(2**256) but however would overflow before that, because the multiplication operation is being called upon on calculation during mintOnBuy() and burnOnSell().

For uint256, it can handle numbers up to 10**76 before overflows, doing a cube of input would overflow when the input is larger than 85 bits (25 decimals places). However, by taking the square of the input and then divide by 10*14, then multiple by the number itself again, although precision is lost, it frees out space that allows calculation of larger input before overflow.

Check on input before calculation to avoid overflow is not implemented as an intent for computational optimization purposes as the case of having over-sufficient decimals in any ERC20 token should be avoided. Described the above condition, the precision should be high enough which would not bring much price impact.

B. Gaping down left-over balance

Since there is a precision loss on each trade, which would round down in mathematical calculation, meaning there would be more balance leftover as trades go on.

By reading the balance of ERC20 the contract has, combined with the supply of Solid-ERC20, the price of each trade could be calculated in such a way that left-over from precision loss is taken care of. So, every trade afterward always takes the arbitrage chance of the left-over balance, this could ensure the left-over balance would not grow overtime.

V. PROTOCOL FEE

The Solid Protocol introduces the **dynamic trading fee** (max capped at 20%), traders pay the dynamic trading fee based on a formula (TF in implementation) on all trades, if feeOn is set to True, 20% from the trading fee would be locked for team_address, hence, the reward pool would only receive 80% from the trading fee instead of 100%.

k_reward_accumulated(κ) is used to store the distribution of the trading fee across the reward pool, given the amount of trading fee as P and the amount of staked balance as φ, whenever a sell occurs, k_reward_accumated is updated as the following:

$$\kappa = \kappa + \frac{P}{\varphi}$$

The total collected fee(T) of individuals could be calculated based on the following given t_1 and t_2 for the time of the transaction and last time fees were collected respectively:

$$T = (\kappa_{t_1} - \kappa_{t_2}) * \varphi$$

All fees are locked in the reward pool, those who staked their Solid-token would then can get the token in the reward pool according to their pool share by claiming.

If feeOn is turned on, 20% of all trading fee would be allocated to the developer team of this project. Reward fee would not be sent on every transaction and must be claimed by the developer team as an intent to minimize computational cost on each transaction.

[1] EAVT uses an interval of 1800 seconds, multiplied by 48 would be 86400 seconds (seconds in a day), however, note that on Ethereum blockchain network blocks updates every ~14 seconds, so theoretically there is a fixed possibility for δ.

[2] Proof by staking means locking liquidity as a way to ensure those supply would not be burnt through the Solid Protocol, which could be different from other terms also with the same name ("Proof by staking")

[3] The computational cost could vary with a different version of Solidity, the one that is implemented here is 0.7.6

[4] If the last mintOnBuy() function is called and afterward the burnOnSell() function triggers an overflow, all funds would be permanently locked up in the pool, but such a situation is unlikely to happen since people would know and sell off before an overflow occurs

REFERENCES

[1] Deepashree, Principles of Economics (For Delhi University B. Com Pass Course), Tata McGraw-Hill Education, 2007, Chapter 5.18.

[2] R. Brent and P. Zimmermann, Modern Computer Arithmetic, American Society of Mechanical Engineers, Feb. 2013. Accessed on: Jan. 13, 2021. [Online]. Available: https://members.loria.fr/PZimmermann/mca/mca-cup-0.5.9.pdf 123-170

DISCLAIMER

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. This post reflects the current opinions of the authors and is not made on behalf of Paradigm or its affiliates and does not necessarily reflect the opinions of Paradigm, its affiliates or individuals associated with Paradigm. The opinions reflected herein are subject to change without being updated.