Bidask Protocol

Bazrov Stepan
bazrovstepan@gmail.com

Malyshev Oleg
malolegmc@gmail.com

Kinelovskiy Nikita
rendseast@gmail.com

October 2024

Abstract

Since the advent of blockchain technology, Decentralized Exchanges (DEXes) have undergone significant evolution, with innovations like Uniswap v3 introducing Concentrated Liquidity Market Maker (CLMM) models. These advancements address the capital inefficiencies that were prevalent in previous iterations, like Uniswap v2. This paper analyzes these developments and introduces a novel system optimized for the complexities presented by The Open Network (TON). Unique challenges such as the lack of atomic transactions in the TON environment compelled us to devise a distributed system of smart contracts to manage liquidity efficiently. Our strategy integrates innovative solutions to these technical challenges, maintaining the liquidity concentration benefits of Uniswap v3 while enhancing them to fully leverage TON's capabilities.

1 Introduction

DEXes have come a long way since the inception of blockchain technology, evolving rapidly to address market demands and inefficiencies. One of the most remarkable advancements is the introduction of Uniswap v3, which addresses the inherent inefficiencies associated with v2 pools. Specifically, Uniswap v2 faced limitations, where liquidity providers (LPs) had to allocate capital across the entire price range of a trading pair. This often resulted in underutilized capital, reducing potential returns for LPs and leading to inefficiencies in the liquidity market.

With the launch of Uniswap v3, the concept of Concentrated Liquidity Market Maker (CLMM) emerged, enabling LPs to concentrate their capital within specific price ranges where they expect the most trading activity. This innovation drastically improved capital efficiency, allowing LPs to generate more fees with less capital. The positive impact was evident almost immediately, as the Total Value Locked (TVL) in v3 pools grew up, demonstrating the market's acceptance and the effectiveness of concentrated liquidity.

Since its inception, v3's concentrated liquidity has become an fundamental aspect of DEX architecture across various blockchains. The increasing TVL in v3 pools underscores its importance and widespread adoption. However, when considering the integration of such advanced mechanisms on unique blockchain platforms like The Open Network (TON), several technical challenges arise. One of the primary obstacles is the absence of atomic transactions, as transactions in TON are composed of asynchronous messages. This presents significant difficulties in ensuring seamless and coordinated updates across multiple smart contracts. Additionally, storing large amounts of data within a single contract's storage is complex, making it problematic to retain all the information related to liquidity bins on one smart contract. To address these challenges, it was necessary to develop a more distributed system of smart contracts that could efficiently manage and synchronize the required data. Despite these hurdles, we has successfully implemented a CLMM-based DEX on TON, effectively overcoming these technical barriers and optimizing the network's unique capabilities.

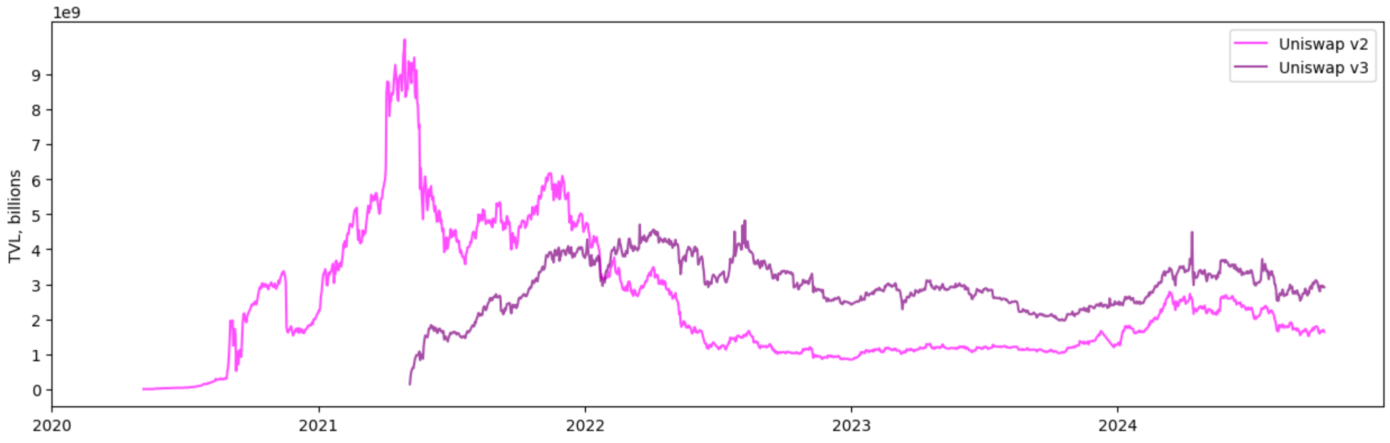


Figure 1: Uniswap v2 and Uniswap v3 TVLs.

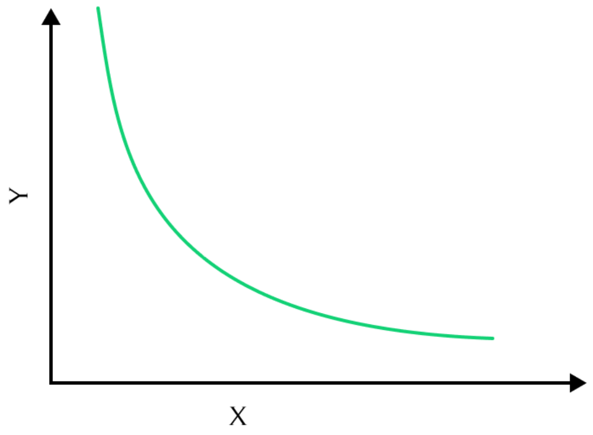
2 Protocol Math

This section describes the mathematics behind Bidask Protocol.

2.1 AMM and bins

The classic AMM in Uniswap v2 operates with a pool where the possible proportions of tokens are described by the constant product formula:

$$xy = k \, [1]$$



Let $P = \frac{y}{x}$. This value is called the *price* and describes how many tokens y you need to give for 1 token x.

Let also $k = L^2$, then $xy = L^2$, and L is the amount of both tokens when the price, or proportion, stabilizes to one (1:1). We call this value liquidity. Liquidity is a property of the pool and it determines the curve of its graph.

In Bidask, the price line is divided into bins by ticks, the positions of which are calculated using the formula:

$$(1+\beta)^i$$

where:

- β – bin step: percentage of price movement between ticks. For example: 0.001.
- i – tick number, $\in (-\infty; +\infty)$.

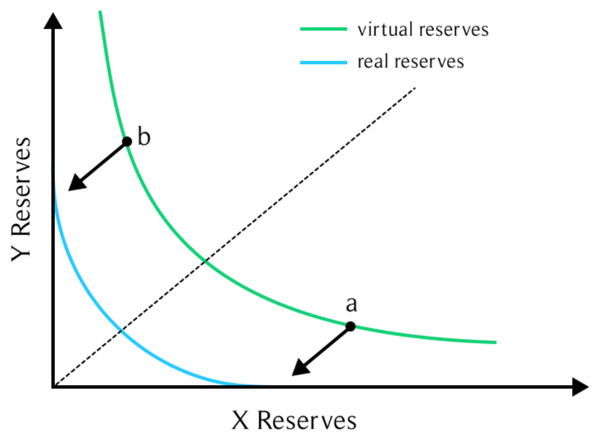
Accordingly, a bin with number i lies within the price bounds:

$$\left[(1+\beta)^i; (1+\beta)^{i+1}\right)$$

Token proportions in bin i are described by the formula:

$$(x + \frac{L}{\sqrt{p_b}})(y + L\sqrt{p_a}) = L^2 \, [2] \qquad (1)$$

This formula "pins" a segment of the constant product curve from price p_a to p_b to the coordinate axes. Thus, on the curve of possible proportions, there are points at which one of the tokens can be completely exhausted (equal to zero).



In it:

- p_a – lower price bound of the bin.
- p_b – upper price bound of the bin.
- L – almost the same as in the classic AMM. The amount of each of the virtual tokens, in case the price is equal to one. The price at one can be outside the bin, but if the segment describing the bin is "moved" back to the full AMM graph, then the necessary (virtual) tokens will be "added" by the x and y coordinates, and the price can be virtually shifted to one.

It is important to note that p_a and p_b here are not necessarily tied to ticks, the formula is universal. Thus, bins essentially "split" one pool into many smaller pools, each described by price bounds and its L.

2.2 Liquidity

Let's learn to find x and y by L (and range bounds) and vice versa based on formula (1). First, it is easy to derive formulas for L of one of the assets, for example:

2.2.1 For x

Here's the basic formula again:

$$(x + \frac{L}{\sqrt{p_b}})(y + L\sqrt{p_a}) = L^2$$

Turn y into 0:

$$(x + \frac{L}{\sqrt{p_b}})L\sqrt{p_a} = L^2$$

Divide both sides by L:

$$(x + \frac{L}{\sqrt{p_b}})\sqrt{p_a} = L$$

Open the brackets:

$$x\sqrt{p_a} + L\frac{\sqrt{p_a}}{\sqrt{p_b}} = L$$

Rearrange:

$$x\sqrt{p_a} = L - L\frac{\sqrt{p_a}}{\sqrt{p_b}}$$

And, finally, express x:

$$x = \frac{L}{\sqrt{p_a}} - \frac{L}{\sqrt{p_b}} = L\frac{\sqrt{p_b} - \sqrt{p_a}}{\sqrt{p_a} \cdot \sqrt{p_b}} \tag{2}$$

Accordingly, L with a given x:

$$L = x\frac{\sqrt{p_a} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{p_a}} \tag{3}$$

This formula calculates what the virtual liquidity is given x in the specified price range, i.e., what the root of the constant product would be if such an amount of x lay in this range in the classic AMM.

2.2.2 For y

Similarly, turn x into 0:

$$\frac{L}{\sqrt{p_b}}(y + L\sqrt{p_a}) = L^2$$

Divide:

$$\frac{y}{\sqrt{p_b}} + L\frac{\sqrt{p_a}}{\sqrt{p_b}} = L$$

And express:

$$y = L(\sqrt{p_b} - \sqrt{p_a}) \tag{4}$$

L with a given y:

$$L = \frac{y}{\sqrt{p_b} - \sqrt{p_a}} \tag{5}$$

2.2.3 With the price inside the range

As mentioned above, the liquidity formula is universal, so it does not matter how liquidity is calculated relative to the bin bounds. Since the calculated liquidity characterizes the AMM curve, the liquidity uniquely determines the amount of the token in some range and vice versa. From this property and formulas (3) and (5), we can derive an equation characteristic of the range in which liquidity is provided while maintaining the constant product (in our case, such a range is any bin):

$$x\frac{\sqrt{P} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{P}} = \frac{y}{\sqrt{P} - \sqrt{p_a}}$$ [3]

When adding a random amount of two tokens in a certain proportion, this proportion is not always maintained. To maintain it, either the missing token must be added, or the excess of the other must be returned. The first option is not suitable, so we need to be able to calculate the excess. The amount of x and y inside the bin is determined relative to each other and the calculated L on them is equal. Knowing this, the excess can be easily found:

$$\Delta L = \min\left(\Delta x \cdot \frac{\sqrt{P} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{P}}, \frac{\Delta y}{\sqrt{P} - \sqrt{p_a}}\right)$$

$$x_{\text{excess}} = \Delta x - \Delta L\frac{\sqrt{p_b} - \sqrt{P}}{\sqrt{P} \cdot \sqrt{p_b}}$$

$$y_{\text{excess}} = \Delta y - \Delta L(\sqrt{P} - \sqrt{p_a})$$

2.2.4 Examples

Since L is the value of one of the tokens at price 1, including x, as the range approaches $[1; +\infty)$, x and L must approach equality. And so:

$$\lim_{[p_a;p_b] \to [1;+\infty)} x$$

$$\lim_{[p_a;p_b] \to [1;+\infty)} \frac{L}{\sqrt{p_a}} - \frac{L}{\sqrt{p_b}}$$

$$\lim_{p_b \to \infty} L - \frac{L}{\sqrt{p_b}} = L$$

Similarly, L approaches equality with y, but in the first half of the entire range – $(0; 1]$:

$$\lim_{[p_a;p_b] \to (0;1]} y$$

$$\lim_{[p_a;p_b] \to (0;1]} L(\sqrt{p_b} - \sqrt{p_a})$$

$$\lim_{p_a \to 0} L(1 - \sqrt{p_a}) = L$$

2.2.5 Minting LP Tokens

The mathematics of minting LP tokens is similar to Uniswap v2:

$$LP_{minted} = \frac{x_{deposited}}{x_{starting}} \cdot LP_{starting}$$ [1]

In essence, LPs are issued according to the deposited share, and thanks to the dependence on the current pool size $x_{starting}$, the "accumulated" fees up to this point remain only in the share of those who previously entered the pool.

2.3 Swapping Process

This section describes the known approaches to the swap algorithm and Bidask Protocol approach.

2.3.1 Uniswap v2 approach

In a simple Uniswap v2 pool, the amounts of tokens X and Y are stored, and the swap algorithm is simple: the fee is taken from the incoming token, a swap occurs with the new amount while keeping the constant product, Y decreases by the required amount sent to the user, and X increases by the full amount sent by the user.

$$\Delta x = (1 - f) \cdot x_{\text{in}} \tag{6}$$

$$\Delta y = y - \frac{L^2}{x + \Delta x}$$

$$(x + x_{\text{in}})(y - \Delta y) = L^2_{\text{new}}$$

Note that when adding the fee to the pool, the proportion is not maintained, as the amount of Y does not decrease. Instead, L increases, which indicates the pool's liquidity.

In the case of Bidask, this direct approach is not applicable because the protocol does not store the real amounts of each bin's tokens, but a similar approach with virtual tokens, described below, is applicable.

2.3.2 Uniswap v3 approach

In Uniswap v3, the amounts of both tokens are not explicitly stored. Instead, the protocol stores the square root of the current price $\sqrt{P}$ and the liquidity L of each bin[1].

The algorithm for finding the amount of tokens to be sent to the user during a swap is as follows: first, it is necessary to find the price movement when buying, for example, X:

$$\Delta\sqrt{P} = \frac{\Delta y}{L} \tag{7}$$

Or the movement of the inverse price when selling X:

$$\Delta\frac{1}{\sqrt{P}} = \frac{\Delta x}{L} \tag{8}$$

Then the opposite delta is calculated:

$$\Delta\frac{1}{\sqrt{P}} = \frac{1}{\sqrt{P} + \Delta\sqrt{P}} - \frac{1}{\sqrt{P}}$$

or, if $\Delta\frac{1}{\sqrt{P}}$ is known:

$$\Delta\sqrt{P} = \frac{1}{\frac{1}{\sqrt{P}} + \Delta\frac{1}{\sqrt{P}}} - \sqrt{P}$$

Then the amount of tokens to be sent to user is found accordingly:

$$\Delta y = \Delta\sqrt{P} \cdot L$$

$$\Delta x = \Delta\frac{1}{\sqrt{P}} \cdot L$$

And the price is updated:

$$\sqrt{P}_{\text{new}} = \sqrt{P} + \Delta\sqrt{P}$$

In Uniswap v3, liquidity provider fees, as well as protocol fees, are calculated similarly to v2 approach (6) and collected separately, so their recompounding into the pool does not require further calculations.

This option for fees accounting is also undesirable, because requires separate storage of earned commissions, which makes auto-recompound impossible, as with simple AMM, when fees are "built-in" into TP-tokens.

2.3.3 Bidask approach

Bidask Protocol implements an algorithm where the price generally only moves as much as the user swaps, including the fee. Thus, the fee is considered in both assets, which slightly reduces volatility and bin change frequency. In this method, it is also unnecessary to find virtual assets.

To perform such a swap, the delta from formulas (7) and (8) must be reduced by the fee amount:

$$\Delta_f\sqrt{P} = \Delta\sqrt{P} \cdot (1 - f)$$

$$\Delta_f\frac{1}{\sqrt{P}} = \frac{1}{\sqrt{P}} \cdot (1 - f)$$

Then the swap is performed as in v3. However, now L needs to be increased by the fee amount using the following formulas:

$$\Delta L = (1 - f) \cdot \frac{\Delta y}{\sqrt{P}}$$

or

$$\Delta L = (1 - f) \cdot \Delta x \cdot \sqrt{P}$$

and

$$L_{\text{new}} = \sqrt{L(\Delta L + L)}$$

2.3.4 Determining Swap Limits in the Bin

Since a bin is a pool with a limited amount of tokens, before the swap, it is necessary to determine if there is available liquidity in the bin. This is achieved by trimming the price: taking the closest of the calculated final price and the bin boundary price.

2.3.5 Slippage

Since TON transaction is not atomic and moving between bins may require multiple messages, a full rollback of the transaction due to exceeding allowable slippage is not possible. Therefore, it is set by the extreme price to which the swap can go. This boundary limits price movement just as the bin boundaries do.

2.3.6 Protocol Fee

Due to the impossibility of rolling back the transaction described above, protocol fee ϕ and referral fee ρ are charged from the returned funds and accumulates separately:

$$y_{\text{out}} = \Delta y \cdot (1 - \phi) \cdot (1 - \rho)$$

2.4 Precision

Although the amounts of tokens are actually integers and cannot take fractional values, it is still necessary to be able to store fractional numbers for price operations. The optimal solution is to set the unit to some power of two, so many auxiliary operations will only require bit shifting. For the fractional part, it is proposed to reserve 128 lower bits of the int type (half). We will call these bits the *precision complement*, and the value $1 \ll 128$ will be denoted as i.

Below is how operations are transformed for correct calculations with such a data type.

2.4.1 Multiplication

When multiplying, the precision complement is considered twice, so it is necessary to divide (in our case, division is equivalent to a right shift) the product by it. In TVM, these two operations can be combined with the `MULRSHIFT` opcode[5], which avoids overflow.

$$(x \cdot y) \ll 128$$

2.4.2 Division and Inversion

When dividing, on the contrary, it is necessary to multiply (shift left) by the precision complement. To avoid overflow, the `LSHIFTDIV` opcode[5] operation is used.

$$\frac{y \ll 128}{x}$$

$$\frac{i \ll 128}{x} = \frac{1 \ll 256}{x}$$

[1]In Uniswap v3, there are no bins per se, but liquidity between ticks is calculated from prefix sums. However, this does not change the essence of the algorithm.

2.4.3 Exponentiation[2]

In integer arithmetic, exponents of different signs must be processed in different branches of the conditional operator. Exponentiation to a positive power is similar to multiplication, but now we need to divide by the precision complement not 1 time, but $n-1$ times:

$$x^n >> (128 \cdot (n-1))$$

In practice, power is calculated through a binary exponentiation algorithm, modified for numbers with precision complement.

As for the negative power, by the definition it is: $x^n = \frac{1}{x^n}$, so it is simply necessary to divide i by the calculated positive power:

$$\frac{i << 128}{x^n >> (128 \cdot (n-1))}$$

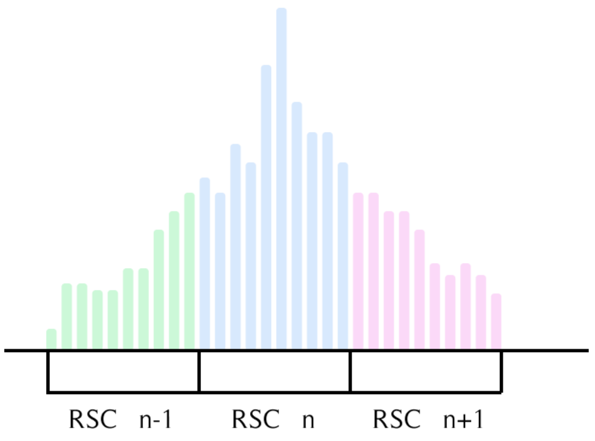
Finally, any number to the power of zero is equal to one, so this branch simply returns i.

2.4.4 Extracting the Square Root

Extracting the square root is implemented using Heron's algorithm, modified for numbers with precision complement and optimized using binary search. The full implementation can be found in the Bidask Protocol code.

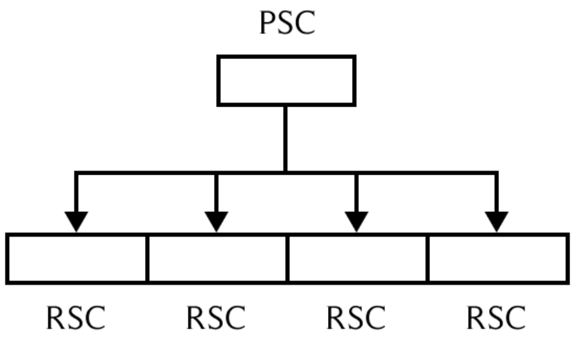
3 Architecture

The key challenge in developing the Bidask Protocol, compared to its counterparts on EVM networks, was the architectural design. TVM discourages the use of large data structures due to the network's focus on sharding and the limits of contract memory associated with this[4]. To address this issue, a Range smart contract (labeled as RSC on a figure) has been implemented, which is responsible for storing partial information about the liquidity of a specific token pair. Essentially, it functions as a sharded array containing information about the corresponding part of liquidity held in this pool. Range also acts as an Automated Market Maker (AMM), handling calculations related to token swaps or deposits into the liquidity pool.

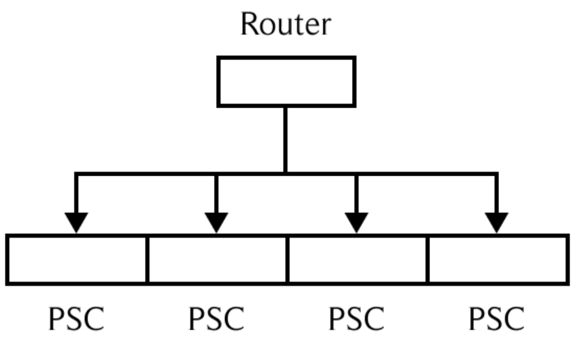


[2]To the power of regular int.

Each range stores a fixed number of bins, allowing for better control over gas expenses when interacting with this contract. All ranges aggregated by the Pool smart contract (labeled as PSC on a figure). This contract is responsible for any interaction with liquidity, directing messages to the appropriate range. In the case of a token exchange, the pool sends a message to the range with the actual bin. In the case of a liquidity deposit, the pool communicates with the range responsible for the bins into which user is adding liquidity.

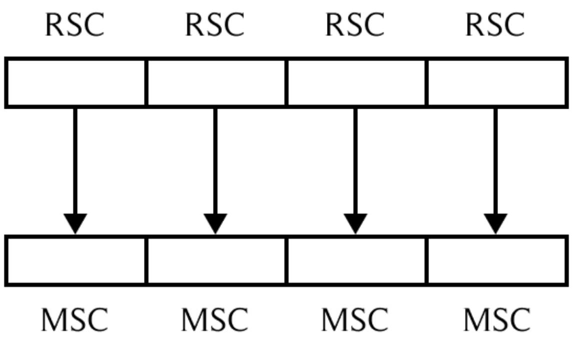


To connect users with the appropriate pool, Bidask architecture incorporates a router smart contract. This contract manages the storage of all tokens, responsible for payouts, and handles routing of both swap and liquidity deposit transactions. While it was originally derived from the STON.fi architecture, it has been adapted to meet new requirements.



3.1 Liquidity position handling

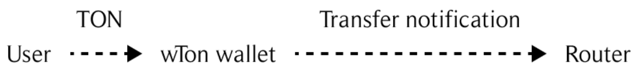
Due to the fragmented nature of liquidity in Bidask, which is divided into multiple bins, a standard token smart contract cannot be used to calculate a user's share of the overall liquidity pool. To address this, a multi-token smart contract (hereinafter referred to as MSC) was implemented. Instead of holding a single number, this multi-token contract maintains an array of values, each representing a user's balance of lp-tokens across different bins.



As a result, each range is paired with a dedicated MSC for each user who deposits liquidity into it. This architecture significantly reduces gas costs for users and allows each range to operate independently, interacting with other ranges only when asset prices change and actual bin shifts.

3.2 Wrapped TON

In addition to the core smart contracts, Bidask architecture includes auxiliary components. The router smart contract can only interact with tokens, so for handling TON, a wrapped TON standart is used. wTON jetton-wallet accepts TON and treats it as a token, enabling seamless operation.



By using this approach, there is no need to account for special cases when a user swaps TON for tokens, allowing the system to operate only through the standard jetton-wallet interface.

4 Conclusion

Bidask Protocol represents a marked evolution in the implementation of CLMM-based DEX architectures within The Open Network. This paper details our successful adaptation of Uniswap v3's mechanisms to accommodate TON's unique infrastructure, where asynchronous message handling posed significant challenges. By addressing these issues, our approach sets a new standard for DEX performance, optimizing both liquidity management and transaction efficiency. As the protocol evolves, it stands as a testament to the ability of blockchain technology to adapt and innovate, ultimately contributing to the broader development of decentralized finance ecosystems.

References

[1] Hayden Adams, Noah Zinsmeister and Dan Robinson. 2020. *Uniswap v2 Core.*

[2] Hayden Adams, River Keefer, Noah Zinsmeister, Dan Robinson and Moody Salem. 2021. *Uniswap v3 Core.*

[3] Atis Elsts. 2021. *Liquidity Math in Uniswap v3.*

[4] Nikolai Durov. 2021. *The Open Network.*

[5] Nikolai Durov. 2020. *Telegram Open Network Virtual Machine.*