Timeswap

First Fully Decentralized AMM based Money Market Protocol

Ricsson Ngo
ricsson@timeswap.io

Harshita Singh
harshita@timeswap.io

Ameeth Devadas
ameeth@timeswap.io

February 2021

Abstract

Timeswap is a *fixed income lending and borrowing protocol* for ERC20 tokens built on Ethereum. It is a non-custodial, immutable, and censorship resistant protocol leveraging the value of network effects. The maturity time of deposits and loans can be set at any time, giving users astounding flexibility to make money markets that fit the time preferences of most people. It is a brutally minimalist, gas efficient, and self-sufficient protocol that works *without the need of price feeds, oracles, and liquidators, which minimizes attack vectors on the protocol***.** It is an ERC20 token agnostic protocol that provides permission-less usage across the entire ERC20 universe. It is especially useful as a money lego to build exotic Defi products that need discrete time preference liquidity in their contracts. While Timeswap can be thought of as bonds in traditional finance, it is a special Defi Primitive as its financial design makes it a unique instrument which isn't possible in the traditional finance world. Just like how the constant product AMM model revolutionized DEXs, Timeswap aims to revolutionize decentralized lending and borrowing along with insurance and derivatives markets.

Introduction

Timeswap protocol users swap between present ERC20 tokens and future ERC20 tokens. It uses a three variable constant product automated market maker algorithm similar to Uniswap for the interest and collateral calculation. Interest amounts to be paid are known upfront at the time of initiating the transaction.

A Timeswap pool has three parameters, an ERC20 as the asset being lent and borrowed, another ERC20 as the collateral, and a fixed maturity time. Users that interact with this pool will follow the fixed maturity time for their deposits and loan.

Lenders interact with a Timeswap pool with assets in ERC20 token A, collateral in ERC20 token B, and a fixed maturity time. After maturity, they receive the principal, plus fixed interest of token A. In the event that some borrowers of the pool default and that lenders

receive a lower number of token A, lenders get claim on the defaulted borrowers' token B collateral, thus protecting lenders from default risk. This gives lenders astounding flexibility and control on the risk vs yield of their deposits, instead of a homogenous risk versus yield for all lenders.

Borrowers interact with the same Timeswap pool that is interacted with by the lenders. They lock token B into the collateral pool, and withdraw token A from the pool. Before maturity, they are required to pay principal plus interest of token A to withdraw their locked collateral. In the event of default or non-payment, their token B collateral will be distributed to lenders.

Liquidity providers initialize Timeswap pools, and add liquidity into these pools. They can be thought of as being both a lender and borrower at the same time. They deposit token A so that there is present capital liquidity for borrowers to receive. They lock token B as collateral with token A as debt received, to back future claims for lenders to receive.

The protocol can be used to lend and borrow any combination of ERC20 tokens. One powerful use case is *debt financing* for projects by using their governance token as collateral to borrow other assets such as DAI or ETH or any other ERC20 token. It offers projects a less dilutive means of fundraising and provides the benefit of leverage compared to equity funding.

Another powerful use case is *reputation-based financing*. One can create an effective reputation system based on some community, free market, or group driven mechanics, like a personal token system, or credit score token system. Social tokens can be looked at as another version of this personal token system. This explicitly casts value into the reputation of individuals, groups, or organizations. These ERC20 tokens can then be used as collateral, to borrow DAI or ETH or any other ERC20 token. This is essentially an under-collateralized loan, which opens up the possibility of unlocking value that was previously not possible.

Another use case is borrowing using *NFTs* or *ERC721 tokens*. Owners fractionalize ERC721 tokens into fungible ERC20 tokens which can be used as collateral. This gives NFT holders the ability to leverage the value of their NFTs, especially for high value NFTs, as they tend to have higher acceptance as a store of value by lenders.

LP tokens from liquidity pools of many different DeFi protocols can be used as collateral which is another powerful use case. *Liquidity financing* gives liquidity providers the opportunity to take advantage of their investment that may not be utilized as much in other DeFi pools.

Just like how Uniswap is especially valuable as an exchange system for the long tail exotic tokens, Timeswap will be valuable as the lending, borrowing, and collateral system for those exotic tokens.

Time Preference Protocol

The Timeswap protocol is a factory contract that creates pair contracts for any pair of ERC20 tokens, one as the asset and the other as the collateral. The factory contract is a permission-less contract that can be used by anyone to create a pair containing their choice of assets.

The factory contract ensures that there are no duplicate pair contracts with the same pair ERC20 tokens parameter. A pair contract contains multiple pools that hold the pair of ERC20 tokens of different maturities, to facilitate lending and borrowing. Each pool has their own native ERC20 Bond token, ERC20 Insurance token, ERC721 Collateralized Debt token, and ERC20 Liquidity token.

Lenders have lower time preference for their tokens and are looking for passive returns with minimal risks. They use the protocol to receive bonds for lending their tokens in a fixed income setting. They also receive insurance in terms of the pool's paired token, which gives claims to the defaulted borrowers' collateral stake. This explicitly gives each lender control over how much default risks he or she wants to take versus the interest rewards he or she will receive.

Borrowers have higher time preference for the tokens. They want the tokens for present usage, short, or for futures trade, thus leveraging their investments. They use this protocol to borrow tokens with fixed interest costs. They lock collateral stakes. The collateral will never be liquidated in the life of the debt. Borrowers have to pay the debt before maturity time to withdraw their collateral stake, if not, their collateral will be distributed to lenders.

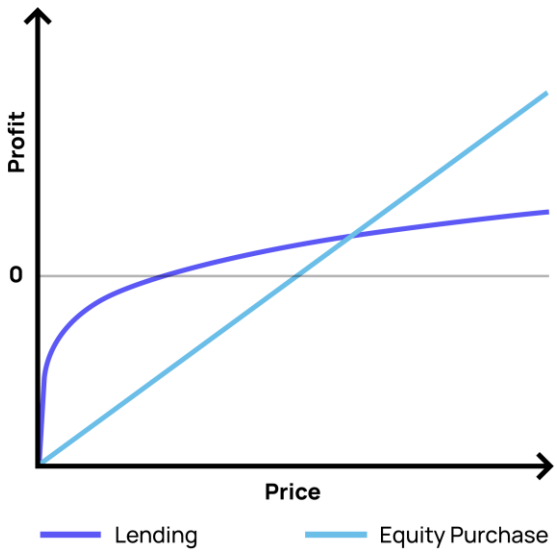
Near maturity time, when the value of the collateral is greater than the value of debt of a collateralized debt, we expect the borrower to pay the debt, but when the value of the collateral is less than the value of debt, we expect them to default. It is this rational behavior why price feeds and oracles are not needed in the protocol. Liquidation happens automatically based on the decisions of each borrower.

Liquidity providers are the market makers of the pool. They add tokens into the pool, and make both lending and borrowing transactions. They earn spread between lenders and borrowers which is based on the number and size of transactions of the pools. They help add liquidity to facilitate transactions between lenders and borrowers.

Comparison to Equity Purchase

The risk vs reward behavior of a direct equity buyer of the ERC20 token versus a lender of a Timeswap pool where the ERC20 token is set as the collateral is shown in the graph

Based on the price of the ERC20 token, the direct equity buyer takes on the full exposure of both the upside and downside. On the other hand, lenders' investment is less risky, as their deposits are backed by a larger value of ERC20 tokens. However, lenders have lower upside due to the interest return.



ERC20 Bond Token

An ERC20 Bond Token contract has an underlying ERC20 token with a fixed maturity time. The recipients of the Bond tokens are the lenders who lent the underlying ERC20 token into the pool. An owner of the Bond tokens can burn it after the maturity, to withdraw the same amount of underlying ERC20 tokens.

The name of the Bond token is the following:

Timeswap Bond - {Name of asset} - {Name of collateral} - {maturity time in Unix timestamp}

The symbol ticker of the Bond token is the following:

TS-BND-{Symbol of asset}-{Symbol of collateral}-{maturity time in Unix timestamp}

For example, consider a user who owns 3000 TS-BND-DAI-WETH-1750000000. Assuming there are enough DAI in the pool after Unix time 1750000000, the owner receives 3000 DAI.

ERC20 Insurance Token

An ERC20 Insurance Token contract has an underlying ERC20 collateral token with a fixed maturity time. The recipients of the Insurance tokens are the lenders who lent the underlying ERC20 collateral token into the pool. For the percent of Bond tokens that the owner doesn't receive, the owner of the Insurance token can burn the Insurance tokens after the maturity to withdraw up to the same percent of the Insurance tokens of the underlying collateral tokens which were staked by the borrowers who defaulted.

The name of the Insurance token is the following:

Timeswap Insurance - {Name of asset} - {Name of collateral} - {maturity time in Unix timestamp}

The symbol ticker of the Insurance token is the following:

TS-INS-{Symbol of asset}-{Symbol of collateral}-{maturity time in Unix timestamp}

For example, suppose the same user owns 10 TS-INS-DAI-WETH-1750000000. Supposedly 20% of Bond Tokens claims are not realized, then the owner can receive up to 2 WETH.

ERC721 Collateralized Debt Token

An ERC721 Collateralized Debt Token contract has an underlying debt in terms of an ERC20 token, an underlying collateral in terms of the paired ERC20 token, and a fixed maturity time. This token is received by each borrower for each of their borrowing transactions. An owner of the collateralized debt token can burn it to withdraw the underlying token set as the collateral after repaying the underlying debt before the maturity.

The name of the Collateralized Debt token is the following:

Timeswap Collateralized Debt - {Name of asset} - {Name of collateral} - {maturity time in Unix timestamp}

The symbol ticker of the Collateralized Debt token is the following:

TS-CDT-{Symbol of asset}-{Symbol of collateral}-{maturity time in Unix timestamp}

For example, there is a user who owns a TS-CDT-DAI-WETH-1750000000 token with 2 ETH collateral and debt of 300 DAI. If the user repays 300 DAI debt before Unix time 1750000000, they can withdraw the 2 ETH collateral. If the user did not pay the 300 DAI debt, the collateral staked by them will be distributed to the lenders and liquidity providers.

ERC20 Liquidity Token

An ERC20 Liquidity Token represents proportional ownership of the pool. The recipients are the liquidity providers who add liquidity into the pool. They can burn their liquidity tokens to withdraw their liquidity from the pool after maturity. The amount received is the proportional amount of all liquidity after all the claims to lenders are realized.

The name of the Liquidity token is the following:

Timeswap Liquidity - {Name of asset} - {Name of collateral} - {maturity time in Unix timestamp}

The symbol ticker of the Liquidity token is the following:

TS-LIQ-{Symbol of asset}-{Symbol of collateral}-{maturity time in Unix timestamp}

For example, there is a user who owns 20% of all TS-LIQ-DAI-WETH-1750000000 tokens. After maturity and after reducing the claims given to lenders, there are 1000 DAI and 4 ETH left in the pool. The user then will receive 200 DAI and 0.8 ETH from the pool. Note that it is possible the liquidity providers will only receive the paired ERC20 token collateral, given that lenders claims took all the ERC20 tokens assets.

Constant Product Automated Market Maker Algorithm

Inspired by the Uniswap protocol, interest and collateral required is set automatically based on the proportion of the reserves in the pools. Depositing tokens and withdrawing tokens changes the reserves of the pool, thus shifting the interest price and collateral required. Lending more tokens into the asset pool decreases the interest rate and collateral required. Borrowing more tokens from the asset pool increases the interest rate and collateral required.

Let K be the *Constant Product*.

Let X be the *Principal Parameter* which is a virtual pool that determines the amount of the assets in the pool that can be borrowed.

Let Y be the *Interest Rate Parameter* which is a virtual pool that determines the interest amount per second.

Let Z be the *Collateralization Pool* which is a virtual pool that determines the insurance claims for lenders and collateral required by borrowers.

The parameters follow the constant product formula given below.

$$XYZ = K$$

Additionally, there are two other pools:

C= Collateral Pool and is equal to the amount of collateral locked by the borrowers

A= Asset Pool and is equal to the amount of assets in the pool. It is the sum of assets lent by the lenders & the debt paid by the borrowers

Borrow Transaction

Let ΔX be the decrease of X pool.

Let ΔY be the increase of Y pool.

Let ΔZ be the increase of Z pool.

The value of ΔX, ΔY, and ΔZ is calculated from maintaining constant K.

$$(X - \Delta X)(Y + \Delta Y)(Z + \Delta Z) = K$$

From observation of the algorithm, as ΔY increases, ΔZ has to decrease; when ΔZ increases, ΔY has to decrease.

The maximum of ΔY is when $\Delta Z = 0$.

$$(X - \Delta X)(Y + \Delta Y_{max})(Z) = K$$

The maximum of ΔZ is when $\Delta Y = 0$.

$$(X - \Delta X)(Y)(Z + \Delta Z_{max}) = K$$

Let d be the duration in seconds from the transaction to the maturity of the pool.

The transaction can only be called before the maturity of the pool.

The borrower receives ΔX assets from the pool.

The borrower will have a debt of $\Delta X + d\Delta Y$ asset ERC20 tokens.

Let $R = \frac{X+dY}{X}$ be the marginal debt to principal ratio for the pool.

The borrower will have to lock $R\Delta Z_{max} + \Delta Z$ collateral ERC20 tokens.

There is a restriction where $\Delta Y \geq \frac{\Delta Y_{max}}{16}$, so there is a minimum interest debt for borrowers.

For the borrowers, the more collateral they lock, the lower interest they have to pay. This reflects in the fact that higher the collateral, lower the probability of borrower default.

Lend Transaction

Let ΔX be the increase of X pool.

Let ΔY be the decrease of Y pool.

Let ΔZ be the decrease of Z pool.

The value of ΔX, ΔY, and ΔZ is calculated from maintaining constant K.

$$(X + \Delta X)(Y - \Delta Y)(Z - \Delta Z) = K$$

From observation of the algorithm, as ΔY increases, ΔZ has to decrease; when ΔZ increases, ΔY has to decrease.

The maximum of ΔY is when $\Delta Z = 0$.

$$(X + \Delta X)(Y - \Delta Y_{max})(Z) = K$$

The maximum of ΔZ is when $\Delta Y = 0$.

$$(X + \Delta X)(Y)(Z - \Delta Z_{max}) = K$$

Let d be the duration in seconds from the transaction to the maturity of the pool.

The transaction can only be called before the maturity of the pool.

The lender deposits ΔX assets into the pool.

The lender will receive $\Delta X + d\Delta Y$ Bond tokens.

Let $R = \frac{X+dY}{X}$ be the marginal bond to principal ratio of the pool

The lender will receive $R\Delta Z_{max} + \Delta Z$ Insurance tokens.

There is a restriction where $\Delta Y \geq \frac{\Delta Y_{max}}{16}$, so there is a minimum interest received by lenders.

For the lenders, the more insurance protection they get, the lower interest claims they receive. This gives each lender flexibility in risk management.

Pay Transaction

Let a be the total asset ERC20 tokens locked in the pool, which are the debt paid back by borrowers and liquidity providers so far.

Let c be the total collateral ERC20 tokens locked in the pool by borrowers and liquidity providers so far.

Let $a*$ be the debt required and let $c*$ be the collateral locked of a collateralized debt token owned by a borrower or liquidity provider.

Let $\Delta a \leq a*$be the debt paid back by the borrower or liquidity provider.

Let $\Delta c \leq c*$be the collateral unlocked by the borrower or liquidity provider.

The transaction can only be called before the maturity of the pool.

The pay transaction follows the proportional equation below.

$$\frac{\Delta a}{a*} = \frac{\Delta c}{c*}$$

The borrower or liquidity provider receives the same proportional amount of collateral as the promotion of debt paid. When the debt is paid fully, the whole collateral is withdrawn.

This makes it so that the state becomes as such below.

$a + \Delta a$ asset locked in the pool.

$c - \Delta c$ collateral locked in the pool.

Withdraw Transaction

Let a be the total asset ERC20 tokens locked in the pool, which are the total debt paid back by borrowers and liquidity providers before maturity.

Let c be the total collateral ERC20 tokens locked in the pool, which are left by borrowers and liquidity providers when they don't pay back the debt of the collateralized debt tokens.

Let b and s be the total supply of bond tokens and insurance tokens owned by all lenders respectively.

Let Δb be the bond tokens and Δs be the insurance tokens owned by a lender respectively.

Let Δa be the asset ERC20 tokens and Δc be the collateral ERC20 tokens received by the lender respectively.

The transaction can only be called after the maturity of the pool.

If the total asset ERC20 tokens in the pool after maturity a is greater than or equal to the total bond b, then the bond token owners will receive is $\Delta a = \Delta b$ asset ERC20 tokens, while the insurance tokens will claim no collateral ERC20 tokens as such $\Delta c = 0$.

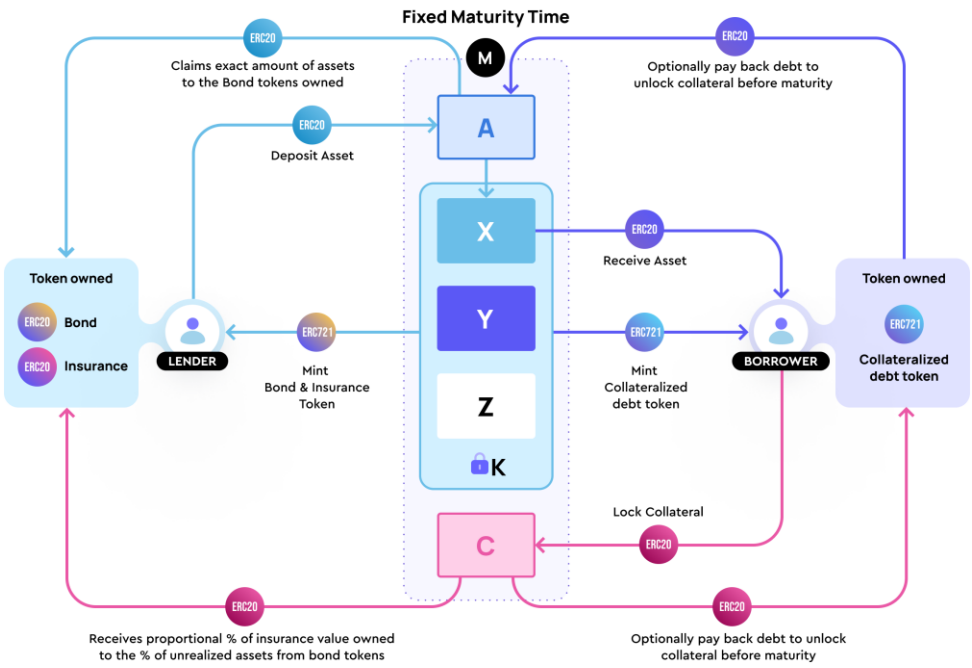
But if a is less than the total bond b, then bond tokens will yield a proportional amount such that.

$$\frac{\Delta a}{a} = \frac{\Delta b}{b}$$

Then the insurance will yield the following collateral amount $\Delta c = (\frac{b-a}{b})\Delta s$, given that there is enough collateral in the pool, $c \geq (\frac{b-a}{b})s$.

If there is not enough collateral in the pool, then the insurance token will yield a proportional amount such that,

$$\frac{\Delta c}{c} = \frac{\Delta s}{s}$$



Adding Subsequent Liquidity Transaction

Let ΔX be the increase of X pool.

Let ΔY be the increase of Y pool.

Let ΔZ be the increase of Z pool.

The value of ΔX, ΔY, and ΔZ is calculated by maintaining the same proportion between X, Y, and Z.

Let $r > 0$ such that $X = r\Delta X$, $Y = r\Delta Y$, and $Z = r\Delta Z$.

$$(X + \Delta X)(Y + \Delta Y)(Z + \Delta Z) = K^*$$

Let d be the duration in seconds from the transaction to the maturity of the pool.

The transaction can only be called before the maturity of the pool.

The liquidity provider deposits ΔX assets into the pool.

The liquidity provider will have a debt of $\Delta X + D\Delta Y$ asset ERC20 tokens.

Let $R = \frac{X+dY}{X}$ be the debt to principal ratio.

The liquidity provider will have to lock $(1 + R)\Delta Z$ collateral ERC20 tokens.

Let l be the total supply of liquidity tokens.

Liquidity tokens Δl will be minted to the liquidity provider that represents proportional ownership of the pool.

$$\frac{\Delta l}{l} = \frac{\Delta X}{X}$$

Adding Initial Liquidity Transaction

Let ΔX be the initial X pool.

Let ΔY be the initial Y pool.

Let ΔZ be the initial Z pool.

Let d be the duration in seconds from the transaction to the maturity of the pool.

The transaction can only be called before the maturity of the pool.

The liquidity provider deposits ΔX assets into the pool.

The liquidity provider will have a debt of $\Delta X + d\Delta Y$ asset ERC20 tokens.

Let $R = \frac{X+dY}{X}$ be the debt to principal ratio.

The liquidity provider will have to lock $(1 + R)\Delta Z$ collateral ERC20 tokens.

The initial Liquidity tokens Δl minted to the liquidity provider is the following:

$$\Delta l = 2^{56}\Delta X$$

The multiple 2^{56} is to make sure that subsequent liquidity providers can add small amount liquidity no more matter how small ΔX and how large ΔY and ΔZ are.

Removing Liquidity Transaction

Let a be the total asset ERC20 tokens left in the pool, which are the total debt paid back by borrowers and liquidity providers before maturity, and subtracted by the claims to lenders.

Let c be the total collateral ERC20 tokens left in the pool, which are left by borrowers and liquidity providers when they don't pay back the debt of the collateralized debt tokens, and subtracted by the claims to lenders.

Let l be the total supply of liquidity tokens owned by all liquidity providers.

Let Δl be the liquidity tokens owned by a liquidity provider.

Let Δa be the asset ERC20 tokens and Δc be the collateral ERC20 tokens received by the liquidity provider respectively.

The transaction can only be called after the maturity of the pool.

The amount of asset ERC20 and collateral ERC20 that the liquidity provider receives is the following.

$$\frac{\Delta a}{a} = \frac{\Delta l}{l}$$

$$\frac{\Delta c}{c} = \frac{\Delta l}{l}$$

Open Market Making

The contract utilizes the open financial market to decide the interest rate of the fixed maturity deposits and loans. It also decides the required minimum collateral stake by borrowers.

When the average interest rate is higher than the market interest rate, or the minimum required collateral by borrowers is higher than the market collateral required (or a combination of both), then it will be a favorable price for the lenders, thus arbitrageur lenders will lend to the pool. Their transactions will increase x, decrease y, and decrease z, thus lowering the average interest rate and minimum required collateral by borrowers.

When the average interest rate is lower than the market interest price, or the minimum required collateral by borrowers is lower than the market collateral required (or a combination of both), then it will be a favorable deal for the borrowers, thus arbitrageur borrowers will borrow from the pool. Their transactions will decrease x, increase y, and increase z, thus increasing the average interest rate and the minimum required collateral by borrowers.

The price changes from transactions of both sides tend to move the average interest rate towards the market average interest rate, and move the required minimum collateral towards the market collateral required.

Optimizing Timeswap Pool Initiation

The market interpretation of the pools X, Y, and Z are useful for arbitrageurs.

The relationship between X and Y is the following.

Let d be the duration in seconds from the transaction to the maturity of the pool.

$\frac{Y}{X}$ is the maximum marginal interest rate per second for the pool. Multiplying by the number of seconds in a year will yield the annual percentage return (APR).

The relationship between X and Z is the following:

$\frac{X}{Z}$ is the minimum marginal collateral factor of the pool.

Slippage Reduction Mechanism

Since Timeswap is fully on-chain, there could be slippage and front running.The receiving amount calculated could change between when the transaction is signed and when it is finalized in a block. Users can bound interest and collateral fluctuations by stating the minimum or maximum amount in the advanced settings. This reverts the transaction if the slippage is too large. They can also set transaction deadlines which revert the orders, if they are not executed fast enough.

A note regarding the slippage of Timeswap transactions compared to Uniswap transactions, the slippage cost from Timeswap is on a different magnitude level to Uniswap. While in Uniswap, we expect that the pair have the same magnitude value in the pool, thus the slippage size is on that same magnitude value. In Timeswap, while the Principal pool is also roughly at the same magnitude size as the Collateral Required pool, the slippage is on the opportunity cost of the collateral locked, not the value of the collateral itself. Also with regards to the Interest Rate pool, the magnitude value of the interest is different to the magnitude value of the principal, therefore the slippage to the interest is also a different magnitude value.

Liquidity Fragmentation

Since anyone can create a pool at any maturity time, it is technically possible to create new pools with the same pair ERC20 for maturity at every second. While it might seem more pragmatic to arbitrarily restrict the number of pools possible to be created every year, a free market process of determining the number of pools per year is a more elegant solution.

A pool with a larger liquidity will provide less slippage, thus a better price to lenders and borrowers. A new pool created at a very near timeframe to an existing pool with a larger liquidity, will be outcompeted, thus incurring loss to the liquidity provider. There is a strong incentive for liquidity to concentrate into one pool rather than be fragmented across multiple pools. If a new pool is created at a far enough timeframe from any existing pool with larger liquidity, then it may stand the chance to survive, as it does not compete with other pools in the same timeframe.

It is expected that there will be multiple local maxima of liquidity pools per pair. The number of pools per pair is expected to be based on the size of the market of that ERC20 pair. Exotic ERC20 tokens that do not have a large market may find that one or two pools per year may be adequate to meet the market demand. However, for high quality ERC20 tokens with a large market, it may be possible to have multiple pools in a year depending on market liquidity.

By relying on the open market behavior of liquidity providers and with the incentives and disincentives naturally found in pool creation and liquidity management, we expect the liquidity providers to create just the right number of pools per pair in any given timeframe.

Transaction Fee

Liquidity providers receive ERC20 Liquidity tokens to represent proportional ownership of the equity of the pool. An important economics that liquidity providers have to take note of is the impermanent loss of their investment. Whenever the proportion of x, y, and z changes from the time they added their liquidity into the pool, the value of their liquidity tokens decreases. This includes the scenario where the lenders took away all the asset ERC20 locked, due to too many borrowers defaulting. When they remove their liquidity given the change of proportions of the pools, they take on that cost.

To provide incentives for liquidity providers to provide liquidity, the contract charges transaction fees to lenders and borrowers. The transaction fee is equal to $_f$ of the interest output and insurance output of a transaction. Lenders will receive $_f$ less interest output & $_f$ less insurance output, thus decreasing the tokens they will receive. Borrowers will mint $_f$ more interest output and lock $_f$ more collateral into the pool, thus increasing the collateral they have to lock and the interest they have to pay. The value of $_f$ is chosen and fixed at the deployment of the factory contract.

Protocol Fee

As a financial reward for the protocol creators for building the protocol, there is a protocol fee per second of p of the liquidity tokens from the liquidity provider when adding liquidity. The total protocol fee is the duration between the initiation of the transaction to the maturity of the pool, multiplied by the protocol fee per second. It also doubles the incentive for the protocol creator to continue building future iterations of Timeswap, as well as for the marketing of the protocol. This protocol fee is chosen and fixed at the deployment of the factory contract; therefore, this percentage value cannot be changed. The protocol fee will be sent to a specific address. The Timeswap team plans to create a Timeswap Decentralized Autonomous Organization to receive the protocol fee and handle the management of the individuals working with Timeswap in a decentralized and censorship-resistant way. The Timeswap Core will stay decentralized & does not provide any special privilege to any user, even to the individuals who are part of the Timeswap DAO. We will follow the motto of "automated in the center, humans at the edges".

References

Adams, H., Zinsmeister, N., & Robinson, D. (2020). *Uniswap v2 Core.* Retrieved from Uniswap.org: https://uniswap.org/whitepaper.pdf

Angeris, G., & Chitra, T. (2020, March). *Improved Price Oracles: Constant Function Market akers.* Retrieved from Arxiv.org: https://arxiv.org/pdf/2003.10001.pdf