Waves.Exchange protocol V1

support@waves.exchange

October 21, 2021

Contents

Waves.Exchange Liquidity Pools

A liquidity pool is a smart contract that enables the automatic trading of its underlying assets and contains assets liquidity, i.e. liquidity pools are a replacement for traditional buyer and seller markets. A liquidity pool contains two tokens - *base* (amountAsset) and *quote* (priceAsset) that are deposited to the pool by liquidity providers.

Anyone can become a liquidity provider for a pool by depositing proportional amounts of the underlying tokens and receiving pool tokens (LP tokens) in return. LP tokens represent liquidity providers' share of the total reserve and can be redeemed for the underlying assets at any time.

Providing liquidity

When a new liquidity provider deposits a pair of tokens to an existing pool, the number of liquidity tokens minted in return is calculated based on the existing quantity of tokens:

$$lpToMint = currentEmissionLP * min(\frac{baseDeposited}{baseInPool}, \frac{quoteDeposited}{quoteInPool})$$

Obviously, the formula above doesn't work for a liquidity provider who is the first to join a pool. In this case, the pool mints LP tokens equal to the geometric mean of the deposited amounts:

$$lpToMint = \sqrt{baseDeposited * quoteDeposited}$$

This also means that the first liquidity provider sets the initial exchange rate by depositing in a proportion that they believe to be an equivalent value between the *base* and *quote* tokens. If this ratio is off the market price, AMM will bring the prices to the equilibrium at the expense of the initial liquidity provider.

All future liquidity providers deposit *base* and *quote* tokens using the exchange rate at the moment of their deposit. If the deposited assets lead to the pool tokens' ratio depegging from the market price, there is a profitable arbitrage opportunity that will be utilized by the AAM, and the price will be corrected.

Working with slippage

Depositing into reserves requires putting both *base* and *quote* tokens according to the current price to the pool smart contract. But, due to the fact that exchanges occur often and are unpredictable, there can be a case when a liquidity provider tries to deposit liquidity to the pool with a price that becomes depegged from the market rate at the time the blockchain receives the transaction. To eliminate that effect, the slippage tolerance feature was introduced.

Before a liquidity provider deposits liquidity to the pool they have to pass the allowable slippage tolerance percentage. It means that in case the ratio of the passed tokens exceeds the pool price for the passed percentage, the transaction will be rejected by the blockchain. Otherwise, the transaction will be accepted.

Under the hood, the contract performs calculations for a fair proportion of the passed token ratio according to the current pool tokens ratio. If the slippage percentage is within the boundaries specified by the user but still exists, the contract calculates fair tokens amounts that will be deposited to the pool. Remaining amounts do not stay intact in the pool contract, but instead are deposited and accumulated in another contract. Later, on a periodic basis, accumulated liquidity is deposited to the pool without emission of LP tokens, thus increasing the value of existing LP tokens.

Removing liquidity

Liquidity providers can send their LP token back to the smart contract at any time to withdraw their proportional share of *base* and *quote* tokens from the pools according to formulas:

$$baseWithdrawn\ =\ baseInPool\ * \frac{lpToBurn}{currentLpEmission}$$

$$quoteWithdrawn\ =\ quoteInPool\ * \frac{lpToBurn}{currentLpEmission}$$

LP tokens are burned. *Base* and *quote* tokens are withdrawn at the current exchange rate (reserve ratio), not the ratio of their original investment. This means some value can be lost from market fluctuations and arbitrage.

Waves.Exchange AMM

The liquidity from all liquidity pools is used for automatic market making (AMM). Special AMM software creates a number of market orders on pool asset pairs utilizing pool liquidity. The more assets there are in a pool, the larger trades it can support. Users no longer need to just trade between buyers/sellers. What users do is trade against the liquidity pool. Each trade leads to an increase of pool liquidity.

Order placement algorithm

Notation:

1. A, B is a quantity of amount and price assets on a liquidity pool,
2. ΔA, ΔB is a quantity of amount and price assets in the order,
3. P is a price of amount assets by price assets,
4. K is an invariant, see formula below,
5. A', B', P', K' etc. are new values after an order execution
6. $\gamma$is a spread,
7. $\delta$is slippage.

Formulas:

1. K = A · B
2. P = $\frac{B}{A}$
3. $p_{ask} = P \cdot (1 + \gamma)$ is a price of ask order
4. $p_{bid} = P \cdot (1 - \gamma)$ is a price of bid order
5. The quantity of amount asset in the order
   $\Delta A_{ask} = A \cdot \frac{\delta}{2 + \delta + \gamma}$
   $\Delta A_{bid} = A \cdot \frac{\delta}{2 - \delta - \gamma}$
6. The quantity of price assets in the order: $\Delta B = \Delta A \cdot p$
7. The criterion of proof: $\begin{cases} \delta < \gamma \cdot (2 - \gamma) \\ \delta > 0 \\ \gamma > 0 \end{cases}$

Orders are placed with a price by a given spread γthat is

$p_{ask} = P \cdot (1 + \gamma)$ is a price of ask order

$p_{bid} = P \cdot (1 - \gamma)$ is a price of bid order

If AMM executes an order completely, the account price P changes by a given slippage δthat is

P'_{ask}=P · (1 + δ)

P'_{bid}=P · (1 - δ)

To do that, we need to determine the right ΔAthat is the quantity of amount assets in the orders.

We have to prove:

$$\Delta A_{ask} = A \cdot \frac{\delta}{2 + \delta + \gamma}$$

$$\Delta A_{bid} = A \cdot \frac{\delta}{2 - \delta - \gamma}$$

First, how will the quantity of assets change after an order? After an ask order it will be

$$A'_{ask} = A - \Delta A_{ask}$$

$$B'_{ask} = B + \Delta B_{ask}$$

We know the P formula, then A·P = B. Let's substitute new amounts after an ask order:

$$A'_{ask} \cdot P'_{ask} = B'_{ask}$$

$$(A - \Delta A_{ask}) \cdot P'_{ask} = B + \Delta B_{ask}$$

Because $\Delta B = \Delta A \cdot p$ by a definition:

$$A \cdot P'_{ask} - \Delta A_{ask} \cdot P'_{ask} = B + \Delta A_{ask} \cdot \rho_{ask}$$

$$\Delta A_{ask} \cdot (P'_{ask} + \rho_{ask}) = A \cdot P'_{ask} - B$$

$$\Delta A_{ask} = \frac{A \cdot P'_{ask} - B}{P'_{ask} + \rho_{ask}}$$

Because $p_{ask} = P \cdot (1 + \gamma)$ and $P'_{ask} = P \cdot (1 + \delta)$by definition and B = A ·P from the definition of the price:

$$\Delta A_{ask} = \frac{A \cdot P \cdot (1 + \delta) - A \cdot P}{P \cdot (1 + \delta) + P \cdot (1 + \gamma)} = \frac{A \cdot P \cdot (1 + \delta - 1)}{P \cdot (2 + \delta + \gamma)} = A \cdot \frac{\delta}{2 + \delta + \gamma}$$

The same for a bid. After a bid order balance will be:

$$A'_{bid} = A + \Delta A_{bid}$$

$$B'_{bid} = B - \Delta B_{bid}$$

So:

$$\begin{aligned}
&A'_{bid} \cdot P'_{bid} = B'_{bid} \\
&(A + \Delta A_{bid}) \cdot P'_{bid} = B - \Delta B_{bid} \\
&A \cdot P'_{bid} + \Delta A_{bid} \cdot P'_{bid} = B - \Delta A_{bid} \cdot \rho_{bid} \\
&\Delta A_{bid} \cdot (P'_{bid} + \rho_{bid}) = B - A \cdot P'_{bid} \\
&\Delta A_{bid} = \frac{B - A \cdot P'_{bid}}{P'_{bid} + \rho_{bid}}
\end{aligned}$$

Because $p_{bid} = P \cdot (1 - \gamma)$ and $P'_{bid} = P \cdot (1 - \delta)$by definition and B = A ·P from the definition of the price:

$$\Delta A_{bid} = \frac{A \cdot P - A \cdot P \cdot (1 - \delta)}{P \cdot (1 - \delta) + P \cdot (1 - \gamma)} = A \cdot \frac{1 - 1 + \delta}{2 - \delta - \gamma} = A \cdot \frac{\delta}{2 - \delta - \gamma}$$

Profit from trades proof

We can conclude that K is always increasing. This means liquidity providers always have some profit from trades. Let's determine the dependency between profit, $\gamma$and δ.

We define profit = K' - K > 0

To have a profit:

$$\begin{cases} \delta < \gamma \cdot (2 - \gamma) \\ \delta > 0 \\ \gamma > 0 \end{cases}$$

Let's show that we need $\delta < \gamma \cdot (\gamma + 2)$to have profit from asks. We have:

$A'_{ask} = A - \Delta A_{ask}$and $B' = B + \Delta B_{ask}$by a definition of ask

$\Delta A_{ask} = A \cdot \frac{\delta}{2+\gamma+\delta}$

$\Delta B_{ask} = \Delta A_{ask} \cdot p_{ask}$by definition

Also, we need to deduce $A'_{ask}$from A, $\gamma$and δ

$$A'_{ask} = A - \Delta A_{ask} = A - A \cdot \frac{\delta}{2 + \gamma + \delta} = A \cdot \left(\frac{2 + \gamma + \delta}{2 + \gamma + \delta} - \frac{\delta}{2 + \gamma + \delta} \right)$$

$$A'_{ask} = A \cdot \left(\frac{2 + \gamma}{2 + \gamma + \delta} \right)$$

Then:

$$K'_{ask} = A'_{ask} \cdot B' = (A - \Delta A_{ask}) \cdot (B + \Delta B_{ask}) = A \cdot B + A \cdot \Delta B_{ask} - \Delta A_{ask} \cdot B - \Delta A_{ask} \cdot \Delta B_{ask}$$
$$K'_{ask} = A \cdot B + \Delta B_{ask} \cdot (A - \Delta A_{ask}) - \Delta A_{ask} \cdot B = A \cdot B + \Delta B_{ask} \cdot A'_{ask} - \Delta A_{ask} \cdot B = A \cdot B + \Delta A_{ask}$$
$$K'_{ask} - K = \Delta A_{ask} \cdot \rho_{ask} \cdot A'_{ask} - \Delta A_{ask} \cdot B > 0$$
$$\rho_{ask} \cdot A'_{ask} - B > 0$$
$$\rho_{ask} = P \cdot (1 + \gamma) \text{ by a definition}$$
$$P \cdot (1 + \gamma) > \frac{B}{A \cdot \left(\frac{2 + \gamma}{2 + \gamma + \delta} \right)} = P \cdot \frac{2 + \gamma + \delta}{2 + \gamma} = P \cdot \left(1 + \frac{\delta}{2 + \gamma} \right)$$
$$1 + \gamma > 1 + \frac{\delta}{2 + \gamma}$$
$$\gamma > \frac{\delta}{2 + \gamma}$$
$$\delta < \gamma \cdot (\gamma + 2)$$

Now we will show that we need $\delta < \gamma \cdot (2 - \gamma)$to have profits from bids. We have:

$A'_{bid} = A + \Delta A_{bid}$and $B' = B + \Delta B_{bid}$by a definition of bid

$\Delta A_{bid} = A \cdot \frac{\delta}{2 - \gamma - \delta}$

$\Delta B_{bid} = \Delta A_{bid} \cdot p_{bid}$by definition

A'_{bid}:

$$A'_{bid} = A + \Delta A_{bid} = A + A \cdot \frac{\delta}{2 - \gamma - \delta} = A \cdot \left(\frac{2 - \gamma - \delta}{2 - \gamma - \delta} + \frac{\delta}{2 - \gamma - \delta} \right)$$
$$A'_{bid} = A \cdot \left(\frac{2 - \gamma}{2 - \gamma - \delta} \right)$$

Then:

$$K'_{bid} = A'_{bid} \cdot B' = (A + \Delta A_{bid}) \cdot (B - \Delta B_{bid}) = A \cdot B - A \cdot \Delta B_{bid} + \Delta A_{bid} \cdot B - \Delta A_{bid} \cdot \Delta B_{bid}$$
$$K'_{bid} = A \cdot B - \Delta B_{bid} \cdot (A + \Delta A_{bid}) + \Delta A_{bid} \cdot B = A \cdot B - \Delta B_{bid} \cdot A'_{bid} + \Delta A_{bid} \cdot B = A \cdot B - \Delta A_{bid} \cdot \rho_{b}$$
$$K'_{bid} - K = -\Delta A_{bid} \cdot \rho_{bid} \cdot A'_{bid} + \Delta A_{bid} \cdot B > 0$$
$$\rho_{bid} \cdot A'_{bid} - B < 0$$
$$\rho_{bid} = P \cdot (1 - \gamma) \text{ by a definition}$$
$$P \cdot (1 - \gamma) < \frac{B}{A \cdot \left(\frac{2 - \gamma}{2 - \gamma - \delta} \right)} = P \cdot \frac{2 - \gamma - \delta}{2 - \gamma} = P \cdot \left(1 - \frac{\delta}{2 - \gamma} \right)$$
$$1 - \gamma < 1 - \frac{\delta}{2 - \gamma}$$
$$\gamma > \frac{\delta}{2 - \gamma}$$

$$\delta < \gamma \cdot (2' - \gamma)$$

So, we have:

$$\begin{cases} \delta < \gamma \cdot (\gamma + 2) \\ \delta < \gamma \cdot (2 - \gamma) \\ \delta > 0 \\ \gamma > 0 \end{cases}$$

Let's make it simpler. To do this, we need to prove, that $\gamma \cdot (2 - \gamma)$always less than $\gamma \cdot (2 + \gamma)$:

$$\gamma \cdot (2 - \gamma) - \gamma \cdot (\gamma + 2) = 2 \cdot \gamma - \gamma^2 - \gamma^2 - 2 \cdot \gamma^2 = -2 \cdot \gamma^2 < 0 \; \forall \gamma > 0$$

So the criterion of proof is:

$$\begin{cases} \delta < \gamma \cdot (2 - \gamma) \\ \delta > 0 \\ \gamma > 0 \end{cases}$$

Contract verification

Besides the AMM formulas, which were designed so that liquidity after each trade increases, separate smart contract verification exists for any order that utilizes pool liquidity. That provides a guarantee that regardless of the market situation, trading volume and price, the contract allows only such orders to be executed which increase the liquidity of the pool's internal assets. This is the basis of base APY that liquidity providers receive automatically.

Base APY calculation

Because our AMM trades always receive a profit, we can estimate BaseAPY by the following formula:

$$APY_{base} = \left(\sqrt{\prod_{\forall\, trades} \frac{A_{after} \cdot B_{after}}{A_{before} \cdot B_{before}}} - 1 \right) \div N_{days} \cdot 365 \cdot 100\%$$

Waves.Exchange token (WX token)

WX is the core token of the Waves.Exchange ecosystem.

WX token inflation

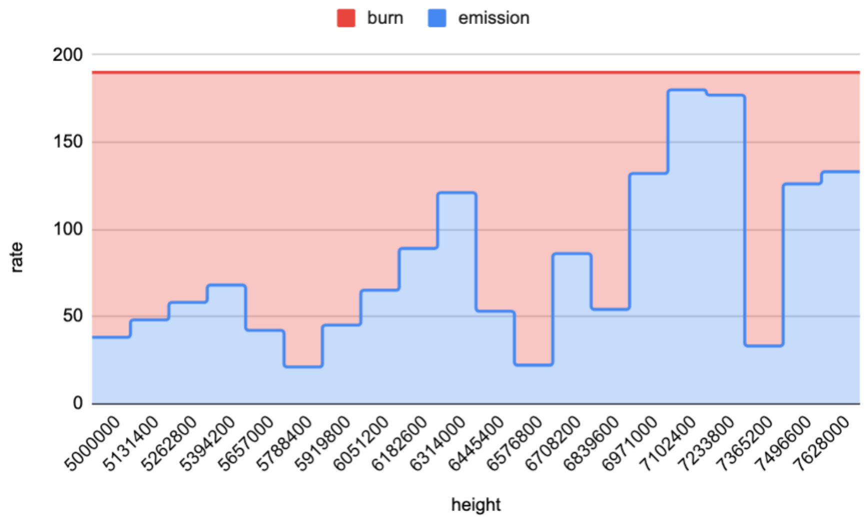
The emitted tokens will be locked in the smart contract and released into circulation in small portions according to emission rate or burned for 2,628,000 blocks (about 5 years).

The release rate indicates the amount of WX released every time a new block is added to the Waves blockchain.

- **ReleaseRateMAX** is a set value of max possible release rate - 190.2587519 WX per block.
- **ReleaseRate** is the current release rate value that is set every 131400 blocks (~ once per quarter of a year) by means of community voting with gWX tokens. ReleaseRate cannot be higher than ReleaseRateMAX.

The difference between ReleaseRateMAX and ReleaseRate is burned by the smart contract.

An example of release (blue area) and burns (red area):



At the same time, ReleaseRate consist of:

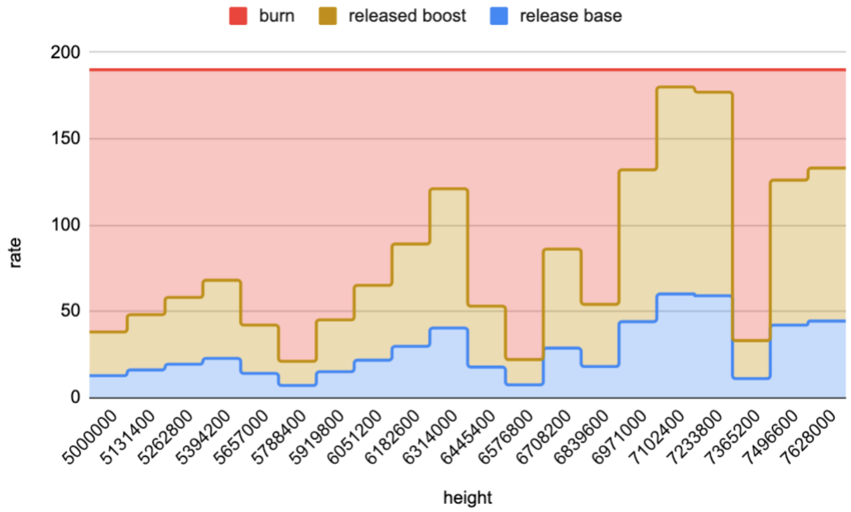
- **ReleaseRateBase** - this rate indicates the amount of WX reward that is distributed among liquidity pools based on their pool weights. Subsequently, the pools distribute the rewards among LP stakers based on their shares. This rate forms the *RewardAPYmin*
- **ReleaseRateBoost** - this rate indicates the amount of WX reward for users that applied boosting to their RewardAPYmin. This rate forms *RewardAPYmax*

The relation between ReleaseRateBase and ReleaseRateBoost is always 1/2, so that:

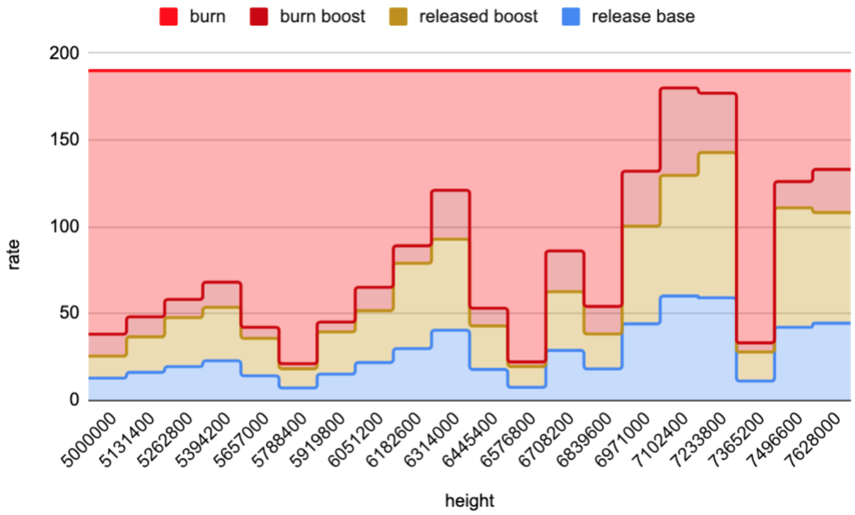
$$ReleaseRateBase \;=\; \frac{1}{3} \times\; ReleaseRate$$

$$ReleaseRateBoost \;=\; \frac{2}{3} \times\; ReleaseRate$$

Then the graph of WX release looks as follows:



Please, note that the ReleaseRateBoost is 100% utilized only if all liquidity providers apply maximum boosting. In reality, this is not likely to happen, so a realistic release graph would look as follows:



WX reward distribution between liquidity pools and users

WX rewards are distributed to liquidity pools based on their pool weights. For the first 129600 blocks (the first quarter of the year) after launch of the liquidity pools, the pool weight w_i for every pool is a predefined constant. A sum of such constants equals 1:

$$\sum_{i}^{n} w_i = 1$$

In the future, the voting feature will be implemented. With this feature, users will be able to change pool weights every 10080 (every week) by using their gWX (gWX explanation can be found below).

Every pool gets WX reward (poolReward) based on its w_i share:

$$poolReward_i = w_i \times ReleaseRateBase$$

LP staking and rewards

A liquidity provider (a holder of LP tokens) can stake their LP tokens to receive a portion of *poolReward* in WX. Rewards can be claimed by the user at any time.

A user’s reward can be calculated at any time like

$$reward_{user}(h) = \frac{\int_{t=0}^{now} b_{user}(h)\, dh}{\int_{0}^{now} B(h)\, dh} \text{ where}$$

$\int_{t=0}^{now} b_{user}(h)\, dh$ – integral of the user's LP balance in staking over height h

$\int_{t=0}^{now} B(h)\, dh = \sum_{i=1}^{n} \int_{t=0}^{now} b_{user}(h)\, dh$, n – number of users

A user can receive $maxReward_{user} = 3 \times reward_{user}$ using gWX and boosting mechanics (see below).

gWX and boosting

gWX explanation

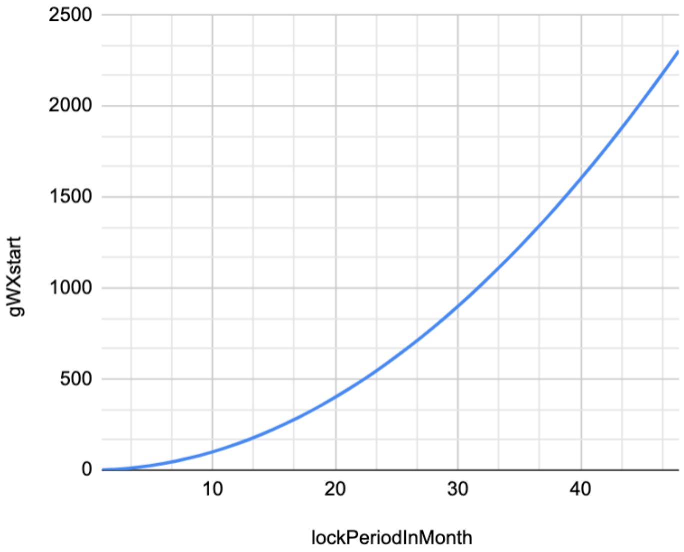
gWX is a unit in which the boosting and the voting power for a particular user is measured. gWX can be received only if you lock WX tokens in a special smart contract for a period from 30240 (~3 weeks) to 2102400 (~4 years) blocks.The locked WX can not be used for standard operations, such as trading, transferring etc. until the end of the selected locking period.

gWX is not a standard token from Waves ecosystem perspective. It's a value of linearly decreasing function y=k*x+b where parameters depend on the amount of locked WX and the locking period t, where $t < t_{max}$ and t_{max} = 2102400 blocks (~4 years). gWX balance gradually decreases starting from the moment when a WX token was locked until it reaches 0 at the end of the selected locking period.

gWX amount at starting height (max value) depends on the amount of locked WX and the duration of the locking period and can be calculated by the following formula:

$$gWXstart = \frac{1}{1000} \times \left(\frac{lockPeriodInBlocks}{30 \times 1440}\right)^2 \times wxAmountForLock$$

Dependency of how many gWX can be received from 1000 WX depending on the duration of the locking period is represented by the following chart:



After receiving gWXstart, we can calculate parameters of linearly decreasing function y=k*x+b:

$$\begin{cases} gWXstart = \ k * lockStartHeight \ + \ b \\ 0 \ = \ k * lockEndHeight \ + \ b \end{cases} \quad \Rightarrow \begin{cases} gWXstart = \ k * lockStartHeight \ + \ b \\ k \ = - \dfrac{b}{lockEndHeight} \end{cases} \quad \Rightarrow$$

$$\Rightarrow \ gWXstart = \ - \frac{b}{lockEndHeight} * lockStartHeight \ + \ b$$

$$\Rightarrow gWXstart = \ b\left(- \frac{lockStartHeight}{lockEndHeight} \ + \ 1\right)$$

$$\Rightarrow gWXstart = \ b\left(\frac{-lockStartHeight \ + \ lockEndHeight}{lockEndHeight}\right)$$

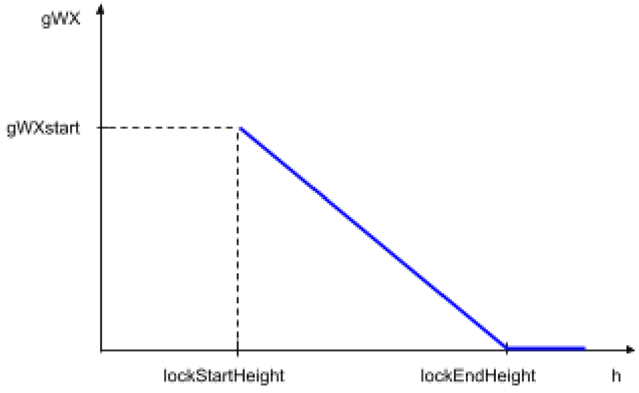
$$\Rightarrow \ b \ = \ \frac{gWXstart * lockEndHeight}{lockEndHeight - lockStartHeight}$$

$$k \ = - \frac{b}{lockEndHeight} \ = \ - \frac{gWXstart}{lockEndHeight - lockStartHeight}$$

As a result, we can build a function depending on blockchain height gWX(h):

$gWX(h) \ = \frac{gWXstart}{lockEndHeight - lockStartHeight} \times \ (lockEndHeight \ - \ h)$, if h < lockEndHeight

$gWX \ = \ 0$,if h < lockEndHeight



gWX is your governance power

Holders of gWX have the power vote in a decentralized manner to affect key decisions regarding:

- WX token RelaseRate
- Liquidity pools weights
- Adding/Removing custom liquidity pools on Waves.Exchange
- Validating custom tokens at Waves.Exchange

gWX rewards (Governance Rewards)

All the collected Waves.Exchange spot trading fees (matching service fees) will be distributed among gWX holders (in proportion to their gWX share).

gWX boosts your LP staking reward

As we explained before, there is a $ReleaseRateBoost$ that determines the amount of WX tokens available for additional distribution among LP stakers (who applied boosting).

gWX is a tool that can be used to automatically boost LP staking reward simultaneously in all the pools. A max boost coefficient cannot be greater than 3.

To calculate a WX reward including boosting ($maxReward_{user}$) for Δh period in $pool_i$we do the following:

$$boostedReward_{user} = reward_{user} + \frac{2}{3} \times poolReward_i \times \frac{\int_{h-\Delta h}^{h} gWX_{user}(h)\, dh}{\sum_{i=1}^{n} \int_{h-\Delta h}^{h} gWX_n(h)\, dh}$$

where

$poolReward_i$- liquidity pool reward, see corresponding section before

n- number of users

$reward_{user}$- user’s reward for LP staking in the pool i

And the final reward cannot be greater than $3\ \times reward_{user}$:

$$maxReward_{user}\ =\ min\,(3\ \times reward_{user},\ boostedReward_{user})$$

This is the Waves.Exchange protocol v1 paper. In the coming weeks, it will be updated to v2, which will bring a detailed governance part. Subsequently, it will get updated to v3, which will add to the paper a complete ecosystem decentralization part. All changes will be announced separately in Waves.Exchange's social media channels.

Disclaimer

This paper is for general information purposes only. It does not constitute investment advice, recommendation or solicitation to buy or sell any investment products and should not be used in the evaluation of the merits of making any investment decisions. It should not be relied upon for accounting, legal or tax advice or investment recommendations. The opinions reflected herein are subject to change without being updated.