Tenderize Whitepaper

Contents

Glossary

Underlying token = original token = token

share = derivative token = tenderToken

SharePrice = effectiveSupply(token)/ effectiveSupply(tenderToken)

targetPrice = sharePrice(we use this term to talk about price we are targeting in the pool => we are targeting sharePrice)

tenderPoolPrice = price of tenderToken in pool

effectiveSupply(tenderToken) = totalSupply(tenderToken) -poolBalance(tenderToken) + mintedForPool(tenderToken)

effectiveSupply(token) = underlyingBalance(token) + underlyingPooled(token) underlyingStaked(token)

Problems of Illiquid staking:

Unstaking period

We want the token to be truly liquid. We use amm pool to achieve this. We keep a liquid portion of the funds in a pool to keep buffer for withdrawals.Thus people do not need to wait the unstaking period.

Liquidifying staking position

If people stake just by providing funds to a regular staker contract, their funds are illiquid. By providing people with tenderToken, we allow them to sell / buy their position freely on secondary market or use it as collateral.

Gas Cost savings

Some of the staking solutions are quite expensive gaswise. Therefore, it does not make sense for people to stake small amounts of funds. That is also the reason to “tenderize” their stake 😊.

Automatic reinvesting of staking rewards

All staking rewards are automatically reinvested; thus we save gas cost and time of our fellow crypto comrades.

System overview

Share calculation

Calculating share price

We need to make sure that share price is always calculated correctly. The goal we need to achieve, is to keep track of all underlying tokens and make sure that the amount of outstanding shares is correct.

We calculate Outstanding share in the system by calling function:
effectiveSupply(tenderToken) = $\mathrm{T}es$

We keep track of all underlying tokens in the system by calling function:
effectiveSupply(token) = $\mathrm{U}es$

Therefore share price = SP is the result of their division

$$SP = \frac{\mathrm{U}es}{\mathrm{T}es}$$

Whenever there is change in amount of tokens we need to make sure that the amount of tenderTokens changes accordingly and vice versa. Hence:

$$\frac{\mathrm{U}es_{new}}{\mathrm{T}es_{new}} = \frac{\mathrm{U}es_{old} + \Delta\mathrm{U}es}{\mathrm{T}es_{old} + \Delta\mathrm{T}es}$$

Staker contract Interactions

Deposit

Alice deposits underlying token (LPT) into the system and gets back tenderToken (tLPT)

1. Token is transferred to Staker contract

2. Based on current shareprice of tenderToken the appropriate amount of tenderTokens are minted and transferred to Alice

3. The original token is staked or used for bringing the pool back to peg:

a. token(whole amount or part of) is traded against pool (this is done if price of tenderToken in pool is equal or higher than share price to keep the peg). The amount received back from the pool is burned to keep the share price same since we have already minted tokens for received funds

b. Token is staked, in our case tenderized 😉

Things to observe:

Since all withdrawals are done by directly swapping tederToken into the pool, we need to rebalance the pool so that the tenderToken price in pool is equal to sharePrice.
We do this by swapping desired amount of underlyingTokens back to the pool and staking the rest.

By this mechanism we ensure that withdrawals do not deplete liquidity in pool. So with every deposit we bring the price in the pool to sharePrice value so people who withdraw can get fair price, even if nobody trades against the pool.

Token change

$$\frac{\Delta \mathrm{U}es}{\Delta \mathrm{T}es} = \frac{\mathrm{deposit(U)}}{minted(\mathrm{T}) + poolOut(\mathrm{T}) \ - burn((poolOut(\mathrm{T}))} = \frac{+minted(\mathrm{T})}{+\mathrm{deposit(U)}}$$

Here we mint the amount of tokens based on shareprice multiplied by deposit. Therefore the share price does not change:

$$\frac{\mathrm{U}es_{new}}{\mathrm{T}es_{new}} = \frac{\mathrm{U}es_{old} + \Delta \mathrm{U}es}{\mathrm{T}es_{old} + \Delta \mathrm{T}es}$$

However, both amounts of underlying tokens and shares increase proportionally.

Withdrawal

Bob withdraws his original token by providing tenderToken

1. tenderToken is transferred to Staker contract
2. Staker swaps whole amount to pool
3. underlyingToken is sent back to Bob

Things to observe:

1. Bob receives his initial deposit + accrued staking rewards + pool rewards – (pool swap fee + slippage)
2. By swapping funds to the pool, tenderPoolPrice becomes lower than targetPrice, however, the price is brought back up in next deposit.
3. By this transaction, the value of tenderToken increases slightly because the amount of underlying token that is sent out of the pool is subtracted by the pool slippage + fee both of which stay in the pool.

   If pool swapped out exactly the amount of tenderTokens times share price, this would not be the case, but since some of the funds stay in the pool as a fee and slippage, the overall amount of tenderTokens outstanding decreases.
   This is because we substract the pool balance when we calculate the amount of outstanding tenderTokens.

Token change

$$\frac{\Delta \mathrm{U}es}{\Delta \mathrm{T}es} = \frac{-\,poolOut(\mathrm{U})}{-\,poolIn(\mathrm{T})}$$

Where $poolOut(\mathrm{U})$ amount can be calculated as such:

$$poolOut(\mathrm{U}) = \ poolIn(\mathrm{T}) * SP \ - poolSlippage(T) * SP \ - poolFee(T) * SP$$

Therefore the amount of underlying tokens that Bob receives is subtracted by $poolFee(T)$ and $poolSlippage(T)$. Since both fees stay in the pool, they are not counted towards outstanding shares. Hence the value of one share in the system increases.

The effect of this is that **all the fees are effectively redistributed tenderToken holders** who benefit!

Pool interactions

We establish liquidity pool to allow everyone to be able to exit his position and get back underlying tokens + rewards without waiting unstaking period.

Swap in underlying token

Anyone can swap in underlying token into the pool to get tenderToken. This makes sense, since there could be a small arbitrage opportunity thanks to withdrawals which take underlying token out of the pool, thus making tenderToken cheaper creating arbitrage for tenderToken.

Token change

$$\frac{\Delta \mathrm{U}es}{\Delta \mathrm{T}es} = \frac{-\,poolOut(\mathrm{U})}{+poolIn(\mathrm{T})}$$

Since we substract pool balance from effectiveSupply of tenderToken, once anyone swaps tenderToken out of pool, it gets counted towards effectiveSupply. Thus this is true:

$$-\ poolBalance(\mathrm{T}) = +\ effectiveSupply(T)$$

$$\frac{\Delta \mathrm{U}es}{\Delta \mathrm{T}es} = \frac{+\,poolIn(\mathrm{U})}{+\ effectiveSupply(T)}$$

Swap in tender token

This is effectively the same as using withdraw().

Token change

Same as withdraw()

Changing pool liquidity

For pool to be usable we need to provide initial liquidity for the pool. As the system grows and there is more daily amounts in deposits / withdrawals we may decide to use some of the staking rewards to increase pool liquidity. This would assure lesser slippage for anyone interacting with the pool.

liquidityIn

Whenever we increase liquidity in the pool we decrease amount of effectiveSupply of tenderTokens since we subtract pool balance from effectiveSupply(see Glossary). For this not to affect share price, we need to add this amount back to the effective supply.
Also nothing changes for underlying token, since we count both Staker balance and Pool balance towards effective supply. We only move them from one location to another.

$$\Delta \mathrm{T}es = -\ poolIn(T) + mintedForPool(T) = 0$$

liquidityOut

Same case as liquidityIn. When we take out liquidity from the pool we need to subtract tenderTokens taken out of the pool, since we take them out of the pool thus they become included in the effective supply.
Nothing changes for underlying token, since we count both Staker balance and Pool balance towards effective supply. We only move them from one location to another.

$$\Delta \mathrm{T}es = +poolOut(T) - mintedForPool(T) = 0$$

Possible “untender” scenarios a.k.a. problems

Run on liquidity

It should be the case that there is net inflow of funds into the system, since as the system grows more people come into the system hence more funds are deposited than withdrawn.

However, in the future it may happen that there are more withdrawals than deposits, thus there is net outflow of funds in the system.

In such a case, the spot price of tenderToken in the pool would move apart from target price making withdrawals more costly. This would however create arbitrage opportunity that anyone can take.

There are couple of approaches how we mitigate the problem.

Arbitrage opportunity to keep the price peg

Firstly by we allow anyone to unstake, by providing tenderTokens to the Staker contract, which will unstake the tokens for the participant and sends him the underlying tokens once the unstaking period has passed.

We do not actually expect people to unstake themselves even though they could, but this incentify arbitragers or anybody able and willing to provide liquidity. They would buy tenderTokens for less than shareprice from the pool, unstake them while hedging for the price movement of the underlying asset during the unstaking period and pocket the difference.

Manual rebalancing

If from whatever reason there are no arbitragers willing to take the arbitrage opportunity, governance can decide to unstake tokens to rebalance the pool.

In practice this would work accordingly:

1. Governance decides to rebalance the pool, if there is too little underlying tokens in the pool and arbitrage does not happen.
2. Previous pool state => share price = 0.8, target price = 1 => 800 / 1000 (token / tenderToken) in pool
3. rebalance to 800 / 800 (this lowers pool liquidity but brings the price back up to target)
4. unstake 200 underlying tokens
5. once unstaked funds are liquid, rebalance pool back to 1000/1000