This proposal details a way of enabling LDO rewards for Curve stETH liquidity providers using the liquidity gauge (still to be deployed, see Curve DAO vote #34

) and a custom rewards contract.

(sorry for the lack of links, new users are not allowed to put more than two of them in a post)

About Curve gauges

Curve's liquidity gauges (source) work in the following way:

1. A user provides the target token(s) into a liquidity pool, getting the LP (liquidity provider) tokens in return.
2. If there's a gauge deployed for that pool, the user can stake their LP tokens into that gauge. As long as LP tokens remain staked in the gauge contract, the user receives their share of the CRV inflation, proportional to the relative weight of the pool among other pools (determined by the Curve DAO on a periodic basis) and to the user's relative stake in the pool's total liquidity.
3. The gauge admin can also assign an external rewards contract that will be called by the gauge each time the total amount of staked LP tokens changes (deposit/withdrawal of LP tokens) and each time someone wants to collect their rewards. This contract may assign reward tokens to the gauge's address in response. If this happens, the gauge claims the tokens and distributes them between the gauge LP stakers, proportionally to their relative stake.

How, exactly, the gauge distributes the rewards? Each time any user's LP tokens stake changes, two things get (effectively) calculated: the amount of rewards that all users together should receive since the previous change, and the share of each user's staked LP tokens within the total pool's stake. Since the gauge knows that the total stake, as well as each user's stake, hasn't changed in between, it can calculate how much rewards should go to each user. The word "effectively" was used above because the gauge doesn't actually run the unbounded loop over all stakers but instead smartly uses partial integration to do the calculation for each user only when that user performs some action.

The user's rewards (if any) are transferred to the user's address each time that specific user deposits or withdraws LP tokens, as well as when the user explicitly requests to collect the rewards. Delayed distribution is not supported.

Adding LDO rewards

The proposal is to deploy the Synthetix StakingRewards

contract forked by Ben Hauser and to propose setting it as the external rewards contract to the stETH/ETH Curve pool's gauge, aiming to fund it with LDO tokens by Lido DAO voting.

The StakingRewards

contract is more complex than it's actually needed for the use with Curve gauges since it supports tracking individual users' rewards — the gauge already does this, as detailed above, so there will be only one "staker" from the rewards contract's point of view: the gauge contract. But this rewards contract seems to have already been deployed in production so it makes more practical sense to use it rather than to write another, simpler, version that will still require careful testing.

The StakingRewards

contract operates by distributing a set amount of reward tokens (LDO in our case) within the set time period. To start a new period, the rewards distribution operator (Lido DAO voting in our case) calls notifyRewardAmount

function on the rewards contract, passing the amount of tokens to be distributed throughout the period being started. The rewards contract then transfers LDO tokens to its balance using ldoToken.transferFrom

, meaning that it should be allowed to spend the set amount of LDO tokens. The duration of reward periods is defined during the rewards contract deploy, but can be changed afterwards given that there's no active rewards period (i.e. no rewards are being issued).

The important detail here is that, ideally, we want the next rewards period to start right after the previous one finished. This can be done by setting the period duration to e.g. a week and managing to finish the DAO voting for the next period before the current period ends. All that's left to be done then is enacting the passed vote.

There's one difficulty, though. The new period may be started even when the current one hasn't finished yet, in which case the undistributed amount of tokens from the current period will be added to the new one, and the new period will start from the present moment. For example, suppose that the contract is configured to use six-day periods, and the current period was started with 100 tokens. If the DAO vote to start the new period was enacted three days within the current one, supplying 200 tokens, then the new period will start immediately, will end in six days, and will distribute 250 tokens: 200 freshly-supplied and 50 leftover tokens from the interrupted period.

Since anyone can enact a passed vote, this can be exploited by liquidity providers who may do that to increase their short-

term rewards. There are two obvious ways to fix this:

1. Modify the StakingRewards

fork to disallow starting a new period before the current one finishes.

1. Deploy an intermediate contract owned by the DAO voting that will do the same instead, and make it the rewards manager (so only that contract may start new reward periods).

I'm leaning toward the first option since it introduces less complexity.

DAO routine operations

Initially, the DAO votes to approve the deployed StakingRewards

contract to spend the amount of unassigned LDO tokens that's enough for, say, several months of rewards.

Each week (or any other period agreed upon) the DAO initiates a vote to start a new reward period and distribute some amount of LDO tokens as LP reward (e.g. to call notifyRewardAmount

on StakingRewards

or on the intermediate contract).

There is a periodic job that tries to enact the passed votes, so the gap between each two consecutive reward periods is minimized.

When the allowance for the rewards contract to spend LDO becomes insufficient to start a new rewards period, the DAO initiates a vote to increase the allowance.