Accounting oracle

info It's advised to readWhat is Lido Oracle mechanism before

Withdrawal stage

As withdrawals on Ethereum are processed asynchronously, the Lido protocol has to have a request-claim process forstETH holders. To ensure the requests are processed in the order they are received, the in-protocol FIFO queue is introduced. Here is an overview of the withdrawals handling process:

1. Request:
2. To withdraw stETH to ether, one sends the withdrawal request to theWithdrawalQueue
3. contract, locking thestETH
4. amount to be withdrawn.
5. Fulfillment:
6. The protocol handles the requests one-by-one, in the order of creation. Once the protocol has enough information to calculate thestETH:ETH
7. redemption rate of the next request and obtains enough Ether to handle it, the request can be finalized: the required amount of ether is reserved and the locked stETH is burned.
8. Claim:
9. The requestor can then claim their ether at any time in the future. ThestETH:ETH
10. redemption rate for each request is determined at the time of its finalization and is the inverse of theETH:stETH
11. staking rate.

note It's important to note that the redemption rate at the finalization step may be lower than the rate at the time of the withdrawal request due to slashing or penalties that have been incurred by the protocol. This means that one may receive less Ether for their stETH than they expected when they originally submitted the request. One can put any number of withdrawal requests in the queue. While there is an upper limit on the size of a particular request, there is no effective limit as the requestor can submit multiple withdrawal requests. There's a minimal request size threshold of100 wei required as well due torounding error issues .

A withdrawal request could be finalized only when the protocol has enough ether to fulfill it completely. Partial fulfillments are not possible, however, one can accomplish similar behavior by splitting a bigger request into a few smaller ones.

For UX reasons, the withdrawal request is transferrable being a non-fungibleERC-721 compatible token.

It is important to note two additional restrictions related to withdrawal requests. Both restrictions serve to mitigate possible attack vectors allowing would-be attackers to effectively lower the protocol's APR and carry fewer penalties/slashing risk thanstETH holders staying in the protocol.

1. Withdrawal requests cannot be canceled.
2. To fulfill a withdrawal request, the Lido protocol potentially has to eject validators. A malicious actor could send a withdrawal request to the queue, wait until the protocol sends ejection requests to the corresponding Node Operators, and cancel the request after that. By repeating this process, the attacker could effectively lower the protocol APR by forcing Lido validators to spend time in the activation queue without accruing rewards. If the withdrawal request can't be canceled, there vulnerability is mitigated. As noted above, making the position in the withdrawal queue transferrable can provide a "fast exit path" for regular stakers via external secondary markets.
3. The redemption rate at which a request is fulfilled cannot be better than the redemption rate on the request creation.
4. Otherwise, there’s an incentive to always keep the stETH in the queue, depositing ether back once it’s redeemable, as this allows to carry lower staking risks without losing rewards. This would also allow a malicious actor to effectively lower the protocol APR. To avoid this, the penalties leading to a negative rebase are accounted for and socialized evenly between stETH holders and withdrawers. Positive rebases could still affect requests in the queue, but only to the point where rebases compensate for previously accrued penalties and don't push the redemption rate higher than it was at the moment of the withdrawal request's creation.

Request finalization

On each report, the oracle decides how many requests to finalize and at what rate. Requests are finalized in the order in which they were created by moving the cursor to the last finalized request. Oracle must take two things into account:

1. Available ether and redemption rate (also called as 'share rate')
2. Safe requests finalization border

Available ether and share rate

The Oracle report has two parts: the report of the number of validators and their total balance and the finalization of requests in theWithdrawalQueue . The finalization of requests requires data from the first part of the report. Therefore, to calculate this part the oracle report is simulated byeth_call tohandleOracleReport in Lido contract, getting share rate and

amount of ether that can be withdrawn fromWithdrawal andExecution Layer Rewards Vaults taking into account the limits.

The structure of the data for simulation:

- reportTimestamp
-
    - the moment of the oracle report calculation, calculated astimestamp = genesis_time + ref_slot * seconds_per_slot
- ;
- timeElapsed
-
    - seconds elapsed since the previous reported ref slot and the simulated one
- clValidators
-
    - number of Lido validators on the Ethereum Consensus Layer
- clBalance
-
    - sum of all Lido validators' balances on the Ethereum Consensus Layer
- withdrawalVaultBalance
-
    - withdrawal vault balance on the Ethereum Execution Layer for the reported block
- elRewardsVaultBalance
-
    - elRewards vault balance on the Ethereum Execution Layer for reported block. Set to "0
- " if try to simulate report in bunker mode
- sharesRequestedToBurn
-
    - gets fromBurner.getSharesRequestedToBurn()
- withdrawalFinalizationBatches
-
    - Set to "[]
- "
- simulatedShareRate
-
    - share rate that was simulated by oracle when the report data created (1e27
- precision). Set to "0
- "

This data is provided to make the call toLido.handleOracleReport() and the following retrieved values are gathered:post_total_pooled_ether andpost_total_shares .

Therefore,share_rate for the withdrawal request finalization can be calculated as follows:

share_rate = post_total_pooled_ether * 10**27 // post_total_shares

Safe requests finalization border

Considering withdrawals, the Lido protocol can be in two states: Turbo and Bunker modes. The turbo mode is a usual state when requests are finalized as fast as possible, while the bunker mode assumes a more sophisticated requests finalization and activated if it's necessary to socialize the penalties and losses. More details in theWithdrawals Landscape doc.

Turbo mode : there is only a single safe requests finalization border that does not allow to finalize requests created close to the reference slot to which the oracle report is performed.

- New requests border (~2 hours by default)

Bunker mode : there are two additional constraints.

The protocol takes into account the impact of negative factors that occurred in a certain period and finalizes requests on which the negative effects have already been socialized. The safe request finalization border is considered to be the earliest of of the following:

- New requests border
- Associated slashing border
- Negative rebase border

Before examining each border, some notations needed for introduction that are used in the graphs below:

New requests border

The border is a constant interval near the reference epoch in which no requests can be finalized:

So can be calculated as:ref_epoch - finalization_default_shift , where:

AndSLOTS_PER_EPOCH = 32 ,SECONDS_PER_SLOT = 12 ,request_timestamp_margin - gets fromOracleReportSanityChecker.oracleReportLimits() .

Associated slashing border

The border represents the latest epoch before the reference slot before which there are no incompleted associated slashings.

In the image above there are 4 slashings on the timeline that start withslashed_epoch and end withwithdrawable_epoch and some points in time: a withrawal request and reference epoch relationship of the slashnings with which to be analyzed.

Completed non-associated

The slashing is non-associated with the withdrawal request since it started and ended before the request was created. It's completed sincewithdrawable_epoch is beforereference_epoch .

Completed associated

The slashing is associated with withdrawal request, since the request is in its boundaries. In this case the slashing is completed sincewithdrawable_epoch is beforereference_epoch , so all possible impact from it is accounted. This slashing should not block the finalization of this request.

Incompleted associated

Slashing is associated with the withdrawal request and is still going on. The impact from it is still incomplete, so such a request cannot be finalized.

Incompleted non-associated

Incompleted but non-associated slashing do not block finalization of the request. The impact of it is still incomplete, nevertheless users are allowed to redeem.

Computation of the border

The border is calculated as the earliestslashed_epoch among all incompleted slashings at the point ofreference_epoch rounded to the start of the closest oracle report frame minusfinalization_default_shift .

Detailed research of associated slashings

Negative rebase border

Bunker mode can be enabled by a negative rebase in case of mass validator penalties. In this case the border is considered the reference slot of the previous oracle report from the moment the Bunker mode was activated -finalization_default_shift .

This border has a maximum length equal to two times the governance reaction time (where governance reaction time is 72 hours).

Border union

The safe border is chosen depending on the protocol mode and is always the longest of all.

def

get_safe_border_epoch (ref_epoch) : is_bunker = detect_bunker_mode ( )

if

not is_bunker : return get_default_requests_border_epoch ( )

negative_rebase_border_epoch

get_negative_rebase_border_epoch ( ) associated_slashings_border_epoch = get_associated_slashings_border_epoch ( )

return

min (negative_rebase_border_epoch , associated_slashings_border_epoch ,)

Finalization

With the amount of available ETH, share rate and safe border, the oracle callsWithdrawalQueue.calculateFinalizationBatches method to get withdrawal finalization batches.

The value of a request after finalization can be:

- nominal
- (when the amount of eth locked for this request are equal to the request's stETH)
- discounted
- (when the amount of eth will be lower, because the protocol share rate dropped before request is finalized, so it will be equal torequest's shares
- *protocol share rate
-)

Batches - array of ending request id. Each batch consist of the requests that all have the share rate below the_maxShareRate or above it (nominal or discounted).

For example, below an example how 14 requests with different share rates will be split into 5 batches by:

| | • • | • • • • • |----------•------ _maxShareRate | • • • • • | • +----------> requestId | 1st| 2nd |3| 4th | 5th | so:

batches = [2, 6 ,7, 10, 14] To start calculation oracle should pass next variables toWithdrawalQueue.calculateFinalizationBatches method:

- maxShareRate
-
    - calculated on previous step as simulatedShareRate, share rate that was simulated by oracle when the report data created
- maxTimestamp
-
    - max timestamp of the request that can be finalized
- maxRequestsPerCall
-
    - max request number that can be processed by the call. Better to be max possible number for EL node to handle before hittingout of gas
- . More this number is less calls it will require to calculate the result
- BatchesCalculationState
-
    - structure that accumulates the state across multiple invokations to overcome gas limits.

struct

BatchesCalculationState

{ /// @notice amount of ether available in the protocol that can be used to finalize withdrawal requests /// Will decrease on each invokation and will be equal to the remainder when calculation is finished /// Should be set before the first invokation uint256 remainingEthBudget ; /// @notice flag that is true if returned state is final and false if more invokations required bool finished ; /// @notice static array to store all the batches ending request id uint256 [MAX_BATCHES_LENGTH] batches ; /// @notice length of the filled part of batches array uint256 batchesLength ; } To start batch calculation oracle should passstate.remainingEthBudget andstate.finished == false and then invoke the functioncalculateFinalizationBatches again with returnedstate until it returns a state withfinished flag set.

Bunker mode

The withdrawals mode a mechanism to protect users who are withdrawing during rare but potentially adverse network conditions, such as mass slashing. The proposed mechanism includes a “turbo mode” for normal operation or low to moderate impact events and a “bunker mode” for significant impact events, which pauses withdrawal requests until the negative consequences are resolved. The solution aims to prevent sophisticated users from exiting earlier in anticipation of a dramatic network- or protocol-wide event.

The “turbo/bunker mode” aims to create a situation where users who remain in the staking pool, users who exit within the frame, and users who exit within the nearest frames are in nearly the same conditions. The “bunker mode” should activate when there is a negative consensus layer rebase or when one is expected to happen in the future, as it would break the balance between users exiting now and those who remain in the staking pool or exit later. Several scenarios should be

taken into consideration as they may cause a negative CL rebase, including mass slashing events, Lido validators being offline for several hours/days, and non-Lido validators being offline. The “bunker mode” is entered in situations such as new or ongoing mass slashing that can cause a negative CL rebase during the slashing resolution period, negative CL rebase in the current frame, and lower than expected CL rebase in the current frame and a negative CL rebase in the end of the frame.

Bunker mode activation

The bunker mode is activated when a negative CL rebase (a decrease in the total amount of staked tokens) is detected in the current frame or is anticipated in the future. CL rebase is used as an indicator of validators' performance, as it provides a better estimate than MEV received during the frame. The conditions for triggering the "bunker mode" are divided into three categories.

Condition 1. New or ongoing mass slashing that can cause a negative CL rebase

The first condition is when there is a new or ongoing mass slashing that may cause a negative CL rebase. The mode is set up when there are as many slashed validators as can cause a negative CL rebase. The protocol switches back to "turbo mode" once the current and future possible penalties from the Lido slashing cannot cause a negative CL rebase.

Condition 2. Negative CL rebase in the current frame

The second condition is when a negative CL rebase is detected in the current frame. The "bunker mode" is activated, and there is a limit on the maximum delay for withdrawal requests finalization that is set to 2 * gov_reaction_time (~6 days) if there are no associated slashings.

Condition 3. Lower than expected CL rebase in the current frame and a negative CL rebase at the end of the frame

The third condition is when there is a lower-than-expected CL rebase in the current frame and a negative CL rebase at the end of the frame. The "bunker mode" is activated when the Oracle detects this condition. The limit on the maximum delay for withdrawal requests finalization is set to 2 * gov_reaction_time + 1 (~7 days) if there are no associated slashings.

For more details, see“Bunker mode”: what it is and how it worksEdit this page Previous Oracle Operator Manual Next Validators Exit Bus