Subsquid Implementation of the Everywhere LST

Dorian (Pierre) Spiegel, Addison Spiegel
addison@thunderhead.xyz

Abstract

Subsquid is a distributed query engine and data lake able to serve more advanced queries at a cheaper cost than traditional RPCs. It's built on Arbitrum and uses the typical structure of bonding tokens to complex infrastructure, making it a perfect network to implement our "everywhere lst" thesis. StSQD is advantageous to traditional staking because of its optimized APR, hassle-free infrastructure management, and accessible DeFi ecosystem.

1 Introduction

The Subsquid network is built on a network of hundreds of workers (currently around 750). These workers are non-trivial to run, require significant amounts of infrastructure and their rewards are based on uptime and worker performance. Additionally, there is a minimum bond to run a worker (roughly $7500 as of August 2024), meaning that if you don't have this much SQD you are forced to delegate. StSqd is a fully managed liquid staking service that maximises rewards whilst making it extremely easy for the user to become part of the pool. From a user perspective, the entire process is as simple as staking SQD and receiving stSQD instantly, and you can view your stSqd balance increase *every block*. Behind the scenes, stSqd uses the same shares and yieldFactor system as Lido and with the added feature of seeing balance increase every block rather than daily, all minted SQD gets almost instantly staked to a diverse and sophisticated operator set, and the entire protocol is non-custodial and is governed by the holders of stSqd themselves.

2 stSQD

The actual LST is based on Lido's stETH, where the balance of all stakers increase over time without having to submit any extrinsic transactions. Both systems use a system of underlying shares and an increasing yield factor ($balance = shares * yieldFactor$) to increase the balance of users. Lido rebases once a day which increases all user balances in one chunk. However, we cannot do the same thing because we allow unstant unstakes, thus allowing for a system where an actor could mint tokens, wait for the rebase, and then

instantly unstake without actually depositing tokens into the protocol that can be bonded to validators. To combat this problem, we update the LST with the rewards earned for the past epoch, and rather than distributing all rewards to users instantly, they linearly earn rewards over the span of the next epoch. Thus ensuring that all users earning rewards are actually contributing to the protocol. Our share calculation is done by

$$yieldFactor = e_{supply} + \frac{rewards * (timestamp - e_{timestamp})}{epochLength}$$

where e is the state at the end of the last epoch.

3 Operators & WorkerManagers

There are two types of yield on Subsquid: worker yield and delegator yield. Rewards are paid out to both parties every epoch, with workers earning 20-30% and delgators earning 10%. In order to be a worker, one has to bond 100,000 $SQD and run worker infrastructure, which is non-trivial. Rewards are paid out to workers based on the utilization rate of the network and liveness and tenure of the worker (how consistently online the worker is). Delegators can bond any amount of tokens to a worker and they earn half the rewards of the worker proportional to their stake.

3.1 Onboarding Operators

One of the most important parts of stSqd is our operator set. It's imperative that operators are of the highest quality and reliability as possible, and we have a few measures in place to make this happen. Actual operators are selected in a two step process. First, operators will apply and Thunderhead will do extensive due-diligence on them and once selected, they will sign an SLA and Thunderhead will submit a governance proposal to add them as an operator. The proposal contains the operator manager address and operator fee. Operators are only rewarded based on a *reward-share system*. Using a reward share system and not a flat USD rate economically incentivizes operators to keep their workers up and rewards high. StSqd holders will vote on this proposal, and if a certain percentage of holders reject the proposal and ability to add the operator, the operator will not be added (see governance section for more detail). Once an operator is added stSqd will deploy them their own controlled WorkerManager.

3.2 Why do we need WorkerManagers?

A caveat of Subsquid is that the pending reward of workers is internal, meaning there is no way to see the independent rewards of workers without observing the balance change by claiming. This means that in order to measure the reward output of different operators, one must have an address solely responsible for the workers of that operator. WorkerManagers are contracts deployed by stSqd

when governance adds a new operator. Each WorkerManager is the staker and manager of all workers that operator runs, and every rebase the WorkerManager claims the rewards of all workers and reports that back to the LST.

3.3 Registering Workers

In order to maximise the percentage of SQD staked we have a semi-automatic staking system. StakeStack is a double ended queue (allows for removal and addition at the front or back) that contains an operator and allowance of number of workers they can stake. At any time an operator can register workers with metadata and peerIds of their infrastructure. The number of workers they can register at any given time is determined by the amount of free SQD tokens and whether there is enough SQD to fill their slot (and previous slots) in the StakeStack.
In addition, if there is ever a situation where we do not have enough workers in the StakeStack to maximise working capital, there is functionality to delegate tokens to any other workers of our choice.

3.4 Deregistering Workers

Unstaking workers is done in two transactions: deregistering and withdrawing. After one deregisters a worker, they have to wait one epoch (which is 100 blocks) before being able to withdraw the bond. When users burn, it will typically get fulfilled by some surplus of SQD thats in the contract. However, if it is a larger burn and there is not enough free SQD to fill it, stSqd will automatically deregister enough workers pulled from UnstakeStake, which is a double ended queue containing the peerId of a worker, to fulfill the burn. Once the epoch ends and the bond is able to be withdrawn, a user can call *redeem* which will withdraw the bond from the deregistered workers and send it to the user.

4 Rebasing & Rewards

Rebasing is simply done by calling a function on the contract with no parameters, which theorically could be done by anybody but it is gated to a rebaser role to mitigate security risks. Rebasing simply claims all delegation rewards and then iterates through all operators and claims rewards from their respective WorkerManager. Each operator has their own fee, which is a percentage of their rewards, and this percentage of earned rewards is set aside to be claimed by that operator and is excluded from rewards earned by holders. There is also a stSqd fee, which is a share taken from the entirety of rewards as a management fee for the protocol. The yield factor calculation is updated with these rewards to increase user balances.

5 Vesting Contracts

The VestingManager is a contract that allows subsquid vesting contracts to stake with stSqd. Vesting contracts cannot interact with stSqd directly because they would be able to transfer stSqd to another address and subvert the vest. The VestingManager allows vesting contracts to mint and burn stSqd, and claim rewards to a third-party address whilst ensuring they can't transfer any other funds out. When a vesting contract calls *mint*, the Manager increases the balance of their address in a a ledger called *staked* and transfers the corresponding amount of stSqd to their vesting contract address. Vesting contracts are only designed to be able to approve SQD to an address, meaning that without extra features the stSqd would be stuck in the vesting contract. This problem can be subverted by creating a permissioned contract only callable to the VestingManager that increases the allowance of the vesting contract spendable by the VestingManager. This allows the stSqd held in the vesting contract to be sent back to the VestingManager to be burnt. When a vesting contract wants to claim rewards they just call *claim* on the VestingManager, which sends $balance + unstaked + staked$ to a beneficiary address of the vesting contract's choice.

6 Governance

Adding operators and upgrading contracts are both parts of stSqd that need some governance for. StSqd will use the same governance system as our other LSTs like stFlip, which is optimistic popular governance. Whenever we want to add a new operator, change params, or upgrade a contract, there will be a proposal that users can vote on with their stSqd. The transaction will be cancelled if stSqd vote to reject the proposal and there is a sufficient number of tokens voting against (reach chorum).