Fast Withdrawals for Low Liquidity Fungible Tokens

Background

A common criticism of optimistic rollups (in contrast to ZK Rollups) is withdrawal latency. Since state transitions of ZK rollups are verified on the EVM, they are accepted instantaneously, allowing users to bridge to L1 as soon batches/proofs are posted on chain (assume batches are posted every L1 block after data availibility issues are solved). On the other hand, since the security of optimistic rollups hinges on a “fraud proof window” elapsing, users must wait for the duration of this window after they initiate a withdrawal. Of course, if they didn’t have to wait the duration of the window, the sequencer could steal all of the assets on the rollup at the cost of their collateral. This is clearly a huge UX issue for optimistic rollups, and ZK rollups love pointing it out :).

A clever solution to fix this issue is the concept of “Fast Withdrawals”. Fast withdrawals, from a high level, work as follows:

- User locks wETH (or any other token) on the L2, signaling their intention to bridge back to the L1
- Liquidity providers for wETH listens for the withdrawal to be recorded on chain and provides the bridged amount to the user (minus a fee)
- After the fraud proof window, the liquidity provider withdraws the users liquidity (profitting the fee from before)

This is cool. Unfortunately, the system dependends on the existence of liquidity providers for bridged tokens, who may not exist for low liquidity fungible tokens and definitely do not exist for NFTs.

Protocol

The BOLT (Bridge for Optimistic rollups transferring Low liquidity Tokens. Quite a stretch for an acronym) bridge is a potential way to solve the low liquidity fungible token bridging problem for optimistic rollups.

Setup

Liquidity providers

The BOLT bridge consists of many liquidity providers (LPs) that are verifying incoming state roots (just like the Fast Withdrawals bridge) to put their highly liquid tokens into bridge pools

. When a liquidity provider puts their tokens (lets use wETH going forward) into a bridge pool for a certain state update, an equal amount of another token, Slash ETH (slETH) is minted to a bridge vault

. Their stake is a “certification” of the correctness of the given state root. The value of slETH is determined as follows:

1. If the state transition the slETH corresponds to is eventually accepted, the slETH is burned and the LP is able to redeem their wETH back.
2. If the state transition the slETH corresponds to is successfully challenged, then the wETH backing the slETH will be slashed and the owner of the slETH will be able to redeem the slashed wETH.

LPs will constantly be pooling for state roots, withdrawing liquidity from older bridge pools and putting it into newer ones as time goes on.

Users

For the following explanations we will use a fictional token, SHAKTI, as an example of a low liquidity token. Let’s go through the initialization process:

- The SHAKTI founders/DAO/whatever decides on a set of L1/L2 liquidity pools from which outsiders can tell the price of SHAKTI in relation to a high liquidity token, say wETH.
- They then provide the addresses of the liquidity pools and the token itself to the bridge
- In response to the information provided, a transaction is dispatched to the L2 that deploys a canonical wSHAKTI ERC20 contract is deployed on the L2.

Bridging from L1 to L2:

- Alice sends SHAKTI to the BOLT bridge on L1
- Alice then proves that they have done so on the L2, and mints themselves the same amount of canonical wSHAKTI on

on the L2

Bridging from L2 to L1 (assuming the sequencer and LPs are honest):

1. Alice burns their wSHAKTI on the L2. Say they burn 10 wSHAKTI
2. The sequencer includes the burn transaction in an L1 batch
3. Alice then proves that they have burned their wSHAKTI on the L2 and does the following actions within a transaction
4. The bridge pool contract, for the state update including her burn transaction reads the price of SHAKTI to ETH from the liquidity pools provided in the initialization step. Say the price is 5 SHAKTI/ETH
5. Alice then buys 200% of the price of their bridged SHAKTI worth of slETH from the bridge vault (remember, this is where slETH is minted to whenever wETH is staked on a state transition) corresponding to the state update including their withdrawal and sends it to a withdrawal vault

on the L1. In this case, the price is 5 SHAKTI/ETH and they are bridging back 10 SHAKTI, so they would create a withdrawal vault with: 200% of (10 SHAKTI)(ETH/5 SHAKTI)(slETH/ETH) = 4 slETH in it.

- After they have created the withdrawal vault, they can remove their proven SHAKTI tokens from the bridge and interact with them freely, happily ever after.
- After the state transition for the slETH in mention has been accepted, LPs claim their share of the principal wETH and the fees Alice paid for the slETH. In addition the slETH is all burned.

A couple notes on this step.

- One cannot “buy” slETH, they can only mint it into withdrawal vaults for a certain price.
- The price of slETH is determined by the bridge pool. I would guess it would be something like 0.0001 ETH/slETH. This is the profit margin of the LPs.

Bridging from L2 to L1 (assuming the sequencer and LPs are malicious):

1. Alice does NOT burn their wSHAKTI on the L2
2. The sequencer posts a state update saying that Alice did burn their wSHAKTI
3. Alice then proves that they have burned their wSHAKTI on the L2 against the invalid state update and completes the same steps detailed in step 3 above.
4. After the state transition for the slETH in mention has been successfully challenged, Alice has her SHAKTI on L1 and has probably transferred it to someone else on L2, effectively double spending her SHAKTI. Making it even sharper, the wSHAKTI on L2 is not backed.
5. This is where the withdrawal vault created in step 3 comes in handy. Now that the state update has been proven invalid, the LPs who provided the slETH in the withdrawal vault are slashed and the slETH turns into wETH

.

1. Assume the price of the SHAKTI to ETH is still 5 SHAKTI/ETH (really it can be anything up to 2x higher than it was during the time of withdrawal). Then whoever holds any wSHAKTI on the L2 can redeem up to the amount of wETH in the withdrawal vault at a price slightly better than the current price, say 4.95 SHAKTI/ETH. The left over wETH is then burned.

Analysis

Ok, now that the protocol has been described, let’s see what it achieves. We assume that the price of the bridged tokens remain the same during the fraud proof window.

If LPs staked on a valid state update, then only honest users will be able to prove withdrawals. They will get their low liquidity tokens instantly for a small fee, and LPs will realize those fees after the state updates are accepted. Yay.

If LPs stake on an invalid state update, they could steal all of the TVL at the bridge (they can engineer the state to reflect that they have burned the entire supply of all tokens at the bridge). Say the TVL is $10 billion dollars. To steal all of the $10 billion from the L2 back to the L1 the LPs would have to lose $20 billion worth of ETH. So, this attack is not cryptoeconomically feasible.

Another variant on this attack would be, the LPs own all $10 billion at the bridge on the L2. They certify an invalid state update, withdraw the $10 billion TVL. Then, after the update is successfully challenged, the LPs withdraw a bit more than $10 billion of the $20 billion slashed ETH. The attacker started out with $10 billion in low liquidity tokens and $20 billion in

wETH, and ended up with the $10 billion in tokens and a little more than $10 billion in wETH. Again, this attack is not cryptoeconomically feasible.

Note that these attacks have varying profits depending on the value of the bridged tokens. To increase tolerance to price movement in these tokens, just increase the ratio of slETH needed to withdraw tokens from the bridge.

Final notes

One cool aspect is that it is peer-to-pool-to-peer. Since all slETH is pooled in bridge pools, collateral is thick. This is in contrast to bridges with routers and peer-to-peer liquidity providers, in which parties involved need to play more active roles and liquidity is thinner.

Another cool aspect is that it can be built permissionlessly. The bridge does not require any fundamental changes to the underlying optimistic rollup's protocol.

So, is this broken? What's bad about it? How can it be improved?