Low Latency Transaction on Plasma

Currently, Plasma MVP requires all child chain pending transactions to be committed and confirmed on the root chain, i.e. Ethereum, before they are considered finalized and can be used by other transactions. That means that Plasma users must wait for at least 16 seconds on average, which is a whole block confirmation time on Ethereum. And it also means that the transaction latency on Plasma will be even higher if not as equal than that on the root chain, which would be a big setback for Plasma adoptions.

Here we propose a solution that child chain pending transactions can be used as inputs of other transactions in the same block, even the block is not yet committed to the root chain. That means pending transactions are considered to be immediatley confirmed once it's included in the child chain block, which results in super low latencies in child chains, and drastically boost the transactions per speed(TPS). This proposal is inspired by David Knott and Kelvin Fichter's "Plasma w/o Confirmations" and Ben Jones and Kelvin Fichter's "More Viable Plasma"

Transaction Inclusion Confirmation

Once the child chain operator receives a new transaction, it validates the transaction, and if everything goes well, the operator promises to include and commit the transaction within N blocks by immediately signs the transaction and returns the future inclusion block number and transaction index in that block to the user, in the form of sign(rlp(blockNumber，txIndex, txMerkleRoot))

. Therefore, the user can use the pending transaction before it is included and committed to the root chain.

Note that because the operator is supposed to immediately return the transaction inclusion confirmation back to user, it is highly recommended for the operators to connect with each other with a very low latency network.

Block Challenge

When a user receives the inclusion confirmation from the operator, and if he trusts the operator in a certain degree, the user can use the confirmed transaction as an input for other transactions. However, if the operator doesn't keep her promises to commit the user's transaction to the root chain in time, the user can challenge the block in the following ways.

Method I

:

The user can submit the proof of his transaction not being included in the promised block on root chain if he has the infomation of the block. If the challenge is accepted, the block is invalidated.

Method II

:

1. The user can submit rlp(blockNumber，txIndex, txMerkleRoot)

and the inclusion confirmation signature from the operator, along with some ETH as a bond to challenge the block denoting as blockNumber

1. Within a certain period, the operator needs to respond to the challenge by submitting her proof that the transaction is included. If the response is invalid or the operator doesn't respond within that period, the block will be invalidated, and the bond goes back to the user. Or the block is safe, and the user loses his bond to the operator.

Block Invalidation Attacks

Because one can challenge to invalidate blocks by submitting transaction inclusion confirmations, the operators can invalidate blocks, which is essentially to cancel some transactions. In order to mitigate that attack, more rules need to be added as follows:

1. Between two child chain blocks, there must be N root chain blocks;
2. The blocks can be challenged must be within the most recent M number of root chain blocks;
3. Once a challenge is successful, besides the challenger gets the token rewards deposited by the operators, some tokens must be burnt by the operators.

Rule 1 and 2 garantee users have enough time to challenge the blocks, while also make sure that history blocks will not be invalidated.

Rule 3 makes sure that even if the operator can challenge a block to make it invalidated, it will get punished by losing some tokens. That way we can prevent the operator from invalidating its own blocks.

Exit Transaction

1. A transaction must be included in the current block before current transaction index plus P;
2. the block number of the transaction input must be lower than the block number of that transaction, or the transaction index of its input must be less or equal to the transaction index of itself minus P;
3. A new block must be committed to the root chain before M-Q, where M is the most recent number of root chain blocks, and Q is a positive integer smaller than M;
4. The block containing the exit transaction must be committed to the root chain before M number of root chain blocks.
5. blockNum denotes the block number of the exit transaction; txIdx denotes the transaction index in that block; InputBlockNum denotes the block number of the input of the exit transaction; and InputTxIdx denotes the transaction index of the input of that transaction; So we get a exit waiting queue like the following:

Order by MAX(inputBlockNum1 * 100000 + inputTxIdx1, inputBlockNum2 * 100000 + inputTxIdx2) ASC, (blockNum * 100000 + txIdx) DESC

And the exit waiting period is 7 days.

1. All inputs must be validated for the exit transaction.
2. The challenge period is 3 days.

Rule 1 and 2 make sure that between the exit transaction and the latest included transaction, there are P number of transactions, but the input of the exit transaction will not be any of P transactions.

Rule 3 makes sure that the operator cannot commit a very old block to the root chain, so that users will have enough time to challenge.

Rule 4 makes sure that the block containing the exit transaction must be a valid block, and cannot be invalidated.

Rule 5 and 6 garantee that valid transactions will always have higher priorities that invalid ones in the exit waiting queue.

Rule 7 makes sure that valid exits that submitted later than invalid ones still have high priorities.

Withhold Attacks

There's always an attack vector on Plasma that the operator can withold all the information of recently committed blocks, which can be a huge problem for Plasma users. Although it is hard to eliminate this attack, rules can be carefully designed to mitigate it as much as possible.

For example, let's say that Alice submitted a transaction when the current transaction Index is A. By then, she must have validated all previous transactions and get all confirm sigs before A. So, if the operator commits an invalid block or withold a block before A, users can challenge that block to invalidate it. And the transaction Alice submitted will not be included after transaction index A+P. So she can exit as follows:

1. Alice tries to exit with the input of the transaction.
2. If the operator challenges the exit within the challenge period, block info is revealed.
3. Alice can then exit with the transaction itself, because the exit transaction has a high priority than the operator's forged invalid one. As long as it's still within the exit period, the operator's invalid exit transaction is still in the exit queue, so the user can exit successfully. If all users follow the rules, they can successfully exit their transactions, which essentially empties out the root chain smart contract.

Limitations and Constraints

It's because low network latency for the operators is assumed, the proposed mechanism may only work for single operator Plasma or low latency consensus, such as Prove of Authority.

Because blocks can be invalidated, even it will most likely not happen, it is still necessary to consider whether the use scenario can bear that risk.