Technical Design of Ledger

draft v0.2.2

FanHui*

January 23, 2021

Contents

*FanHui, hui@findora.org

1 Background

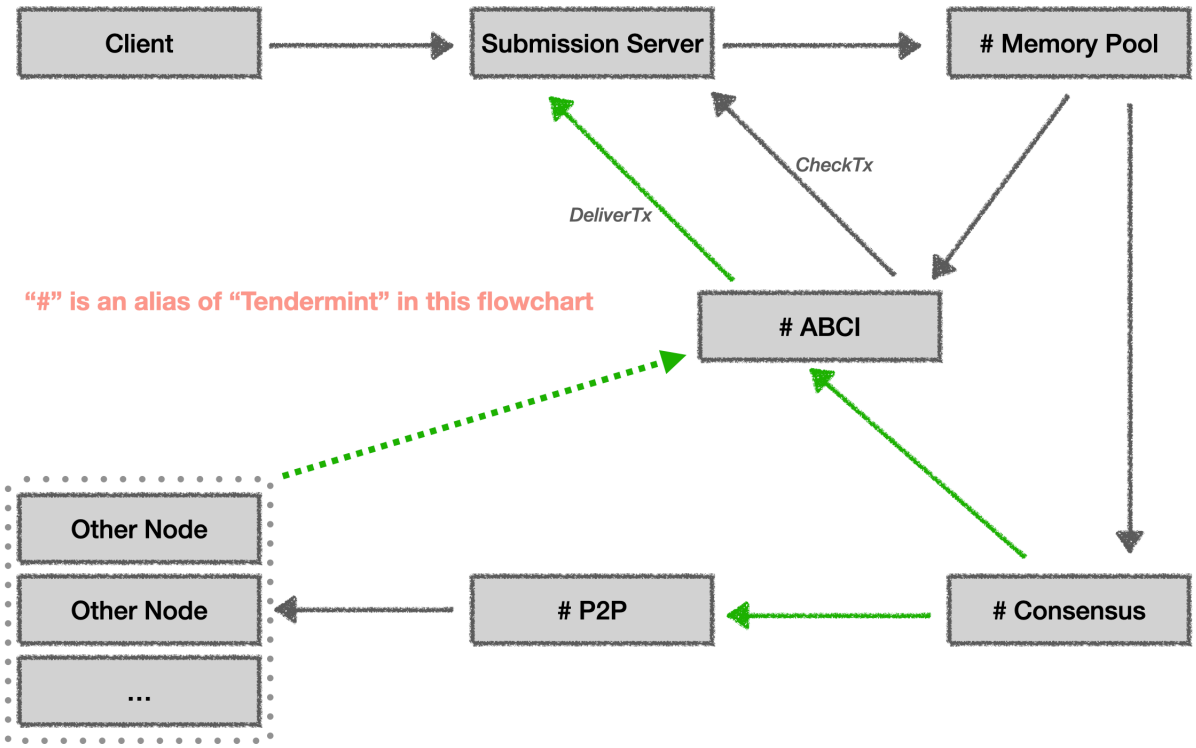

Figure 1: Transactions in Tendermint

Our previous ideas were limited to those blackbox-like operations on the api provided by Tendermint, this brought many limitations to our action space. The solutions described in this article no longer use the premise of completely basing on the tendermint API, and will make changes to the tendermint itself.

The above description may mislead you into thinking that this will lead to more code changes, but this is not the case, your intuition is wrong.

In fact, this strategy will bring a least damage to the existing code. When you read the entire content of this article, you will see that this is almost entirely under the existing framework. On the contrary, if we don't make a few changes to "Tendermint", we will be forced to make a lot of ugly changes to our code, which may cause more serious problems.

Currently, only three parts are covered:

- transaction fee
- genesis state
- block reward

More content may be added in the future, the source code of this article is at FGR (click here).

2 Transaction Fee

2.1 Overview

When the client generates a transaction, the receiving account of the transaction fee is a fixed address, which is completely public, including its private key. Therefore, the client does not need a special method to set transaction fees, and all fees transferred to this address are regarded as transaction fees.

In the following content, FFA (Findora Fee Address) will be used to represent this address.

When the validator generates a block, it can transfer all assets under the FFA to its own account based on the FFA's private key as the block's revenue, usually FRA tokens.

The calculation rule of transaction fee is another issue that needs to be considered, such as charging according to the total number of bytes of the transaction itself, etc. This will be implemented as a method of `Transaction` in the code, which will be referred to in the form of `tx.calculate_fee()` later.

The transaction fee collection logic will be implemented in this way.

2.2 Safety Analysis

1. In a fully public FFA account, anyone can construct a legal transaction signature. How to ensure that the tokens are only obtained by the validator?
   - Analyzing from the economic model, validator will certainly not allow the theft of its assets.
   - The block is packaged by the validator. In the process of packaging, all records of transfers from the FFA account that are not issued by itself will be filtered out.
   - The logic of this part will be implemented by changing the code of Tendermint.
2. Does validator have a chance to create double-spending phenomenon?
   - This needs to be controlled by the business logic of the APP, such as Nonce, etc.
   - The transaction fee mechanism itself does not introduce additional double spending.

2.3 Changes to Findora

- Provide a function to the client to calculate transaction fees during the transaction construction phase.
- Ensure that at least one `Operation` of type `TransferAsset` exists in each transaction, and the token type of operation is FRA.
- According to the result of `tx.calculate_fee()` to calculate whether the minimum transaction fee is met.

```
pub struct Transaction {
    pub body: TransactionBody,
    ...
}

pub struct TransactionBody {
    // Some `Operation`s in the `TransferAsset` type must exist,
    // and the destination of their FRAs must be the `FFA`.
    pub operations: Vec<Operation>,
    ...
}

pub enum Operation {
    TransferAsset(TransferAsset),
    ...
}
```

2.4 Changes to Tenderimint

Create a `golang package` to create transactions for charging transaction fees when `Tendermint` generates a block. It will has an import-path similar to `github.com/FindoraNetwork/findora`, wrapped the `txn_builder` inside it, specifically, we will compile rust code into a static library, and call it through golang FFI, if you do not familiar with this way, see filecoin-ffi.

The comment section in the following codes describes the logic that will be added to `Tendermint`:

```
/**
 * tendermint(v0.33.5)/state/execution.go#92
 */
func (blockExec *BlockExecutor) CreateProposalBlock(
    height int64,
    state State, commit *tGypes.Commit,
    proposerAddr []byte,
) (*types.Block, *types.PartSet) {
    maxBytes := state.ConsensusParams.Block.MaxBytes
    maxGas := state.ConsensusParams.Block.MaxGas

    maxNumEvidence, _ := types.MaxEvidencePerBlock(maxBytes)
    evidence := blockExec.evpool.PendingEvidence(maxNumEvidence)

    maxDataBytes := types.MaxDataBytes(maxBytes, state.Validators.Size(),
        len(evidence))
    txs := blockExec.mempool.ReapMaxBytesMaxGas(maxDataBytes, maxGas)

    // Right here!
    //
    // Filter out transactions that want to steal fees,
    // drop all txs with any input from the `FFA`.
    //
    // Should this be implemented in the ABCI's `CheckTx(...)`?
    //
    // ```
    // txs = findora.FilterThieves(txs)
    // ```
    //
    // After all `txs` from the mempool have been handled correctly,
    // the validator can create a new tx which will transfer all fees
    // from the `FFA` to its own, and append it to `txs`.
    //
    // ```
    // txs = findora.AppendFeeTx(txs)
    // ```

    return state.MakeBlock(height, txs, commit, evidence, proposerAddr)
}
```

3 Genesis State

3.1 Overview

Whether it is by adding `AppState`[1] in `genesis.json`, or the way of sending transactions directly on the client side, there has no guarantee that all the preset transactions will be packaged into the genesis block[2]. The reason is that all transactions sent from outside need to go through tendermint's `MemPool`, The time when these transactions in `MemPool` are confirmed[3] cannot be accurately predicted.

If we can add these preset transactions after `MemPool`, we will get a definite result, that is, all preset transactions will appear in the genesis block.

The transaction process in the genesis block is as follows:

1. Create an FRA asset type with a fixed address[4]. This address will be referred to as FGA (Findora Genesis Address) [5] in the following content,
2. Issue a large number of tokens, such as the maximum value of u64.
3. Create a transaction, send the corresponding amount of tokens to those pre-registered address.
4. These transactions will be obtained by the APP through the regular `DeliverTx` callback, so no additional data synchronization operations are required.

3.2 Safety Analysis

1. Why issue such a large number of tokens?
   - It can be issued on demand.
   - The currency issuance strategy here is an alternative.
2. Why does FGA need to be fully disclosed?
   - If the private key is kept secret, it will bring a crisis of public trust.
   - System will ensure that this address is used correctly through a consensus mechanism.
3. Why do I need to use FFA when transferring money to users?
   - FGA will be restricted to only allow transfers to FFA in the logic of the APP layer.
   - This attribute will also be used by the block rewards in the 4 chapter.

[1] and then sending the transaction accordingly in the APP

[2] Tendermint's genesis block height is 1

[3] aka 'packaged by validator'

[4] this address is similar to FFA and is also fully public

[5] Similar to the concept of coinbase in other blockchains

3.3 Changes to Tenderimint

When the block height is 1, it means that it is in the process of publishing the genesis block.

Similar to the previous section, the `golang package` named `findora` is also required, The function of `txn_builder` is internally encapsulated and used to generate all genesis transactions.

```
/**
 * tendermint(v0.33.5)/state/state.go#131
 */
func (state State) MakeBlock(
    height int64,
    txs []types.Tx,
    commit *types.Commit,
    evidence []types.Evidence,
    proposerAddress []byte,
) (*types.Block, *types.PartSet) {
    block := types.MakeBlock(height, txs, commit, evidence)

    var timestamp time.Time
    if height == 1 {
        // Right here!
        // Write all transactions that need to be in the genesis block.
        //
        // ```
        // txs = findora.GenesisTxs()
        // block = types.MakeBlock(height, txs, commit, evidence)
        // ```
        timestamp = state.LastBlockTime
    } else {
        timestamp = MedianTime(commit, state.LastValidators)
    }

    block.Header.Populate(
        state.Version.Consensus, state.ChainID,
        timestamp, state.LastBlockID,
        state.Validators.Hash(), state.NextValidators.Hash(),
        state.ConsensusParams.Hash(), state.AppHash, state.LastResultsHash,
        proposerAddress,
    )

    return block, block.MakePartSet(types.BlockPartSizeBytes)
}
```

4 Block Reward

4.1 Overview

All `Block Rewards` finally come from FGA.

There can only be one transaction transferred from FGA in each block, and the amount cannot exceed the officially defined block-reward, and its destination must be FFA.

4.2 Changes to Findora

Ensure that the following rules are all effective:

- FGA can only transfer funds to FFA
- FGA can only transfer outward once per block
- Each transfer amount of FGA must not be higher than the officially defined block reward
- A single bad validator cannot reach a consensus on the whole network

4.3 Changes to Tenderimint

Recalling the `CreateProposalBlock()` function in Tendermint that we mentioned in 2.4, we can go a step further, as follows:

```
/**
 * tendermint(v0.33.5)/state/execution.go#92
 */
func (blockExec *BlockExecutor) CreateProposalBlock(
    height int64,
    state State, commit *types.Commit,
    proposerAddr []byte,
) (*types.Block, *types.PartSet) {
    maxBytes := state.ConsensusParams.Block.MaxBytes
    maxGas := state.ConsensusParams.Block.MaxGas

    maxNumEvidence, _ := types.MaxEvidencePerBlock(maxBytes)
    evidence := blockExec.evpool.PendingEvidence(maxNumEvidence)

    maxDataBytes := types.MaxDataBytes(maxBytes, state.Validators.Size(),
        len(evidence))
    txs := blockExec.mempool.ReapMaxBytesMaxGas(maxDataBytes, maxGas)

    // Right here!
    //
    // Below is the form of `AppendFeeTx()`' in the 'Transaction Fee'
        section:
    //
    // ```
    // txs = findora.AppendFeeTx(txs)
    // ```
    //
    // Now, suppose our block reward is 10 token, consider the following
        rules:
    //
    // - transfer 10 token from FGA to FFA
    // - transfer all transactions in FFA to validator
    // - the input address must be the address of `FFA`
    // - transfer from `FGA` and `FFA` is only allowed once in each block
    //     - prevent verifiers from doing bad things
    //     - ensure this in the ABCI's `DeliverTx()` based on our APP logic
    //
    // That's it!
    // The `transaction fee` and `block rewards` will be done at the same
        time.
    // Finally, we rename the original `AppendFeeTx()` to `
        AppendFeeAndRewards()`:
    //
    // ```
    // txs = findora.AppendFeeAndRewards(txs)
    // ```
    //
    // Should the `AppendFeeAndRewards` be splitted to two separated
        functions?
    //     - findora.AppendFee(txs)
    //     - findora.AppendRewards(txs)

    return state.MakeBlock(height, txs, commit, evidence, proposerAddr)
}
```

5 Investment Return

5.1 Overview

Some thoughts about the design of the return on investment:

- We will assume that the token will increase in value over time, which is a natural benefit.
- During mainnet v1.0, all validators may be operated internally, but this does not mean that the income receiving address of the code validator must also be internal. It can be the address of the investor, and a single validator can have multiple such addresses. The weight of each address is allocated based on the investment amount. In general, validator can perform the role of a revenue distribution agent.
- When issuing FRAs, we issue such an investment incentive strategy: at a predetermined block height, we will add up all the transaction fees before this (maybe use other better algorithms instead of simple addition), Issuance of the same amount of tokens as investment income. The method of distribution is: the weight is calculated according to the FRA holding time of all investors and the holding amount at that time, and the final return that each investor will get is determined according to this weight. The principle of this model is: the more transactions, the better the operating status of Findora, and the better the operating status of Findora, the greater the expected return of investors, which in turn will give investors the motivation to do more things that benefit Findora.

6 Summary

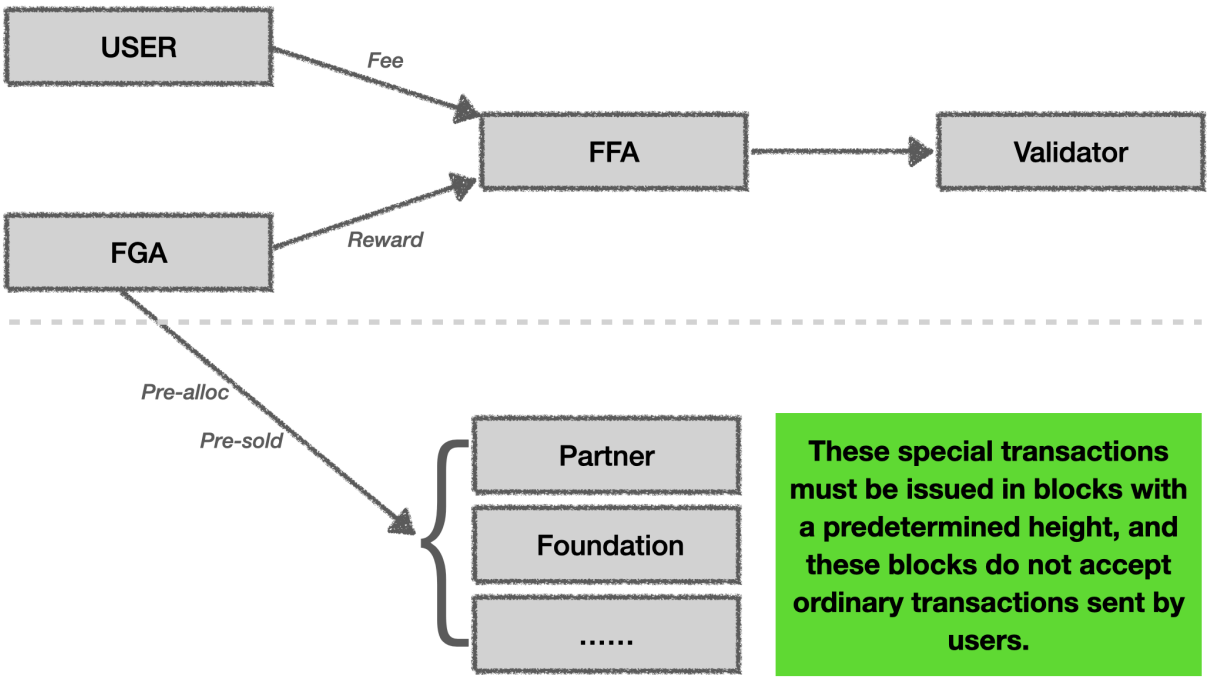

Figure 2: Findora Asset Management

Finally, organize all necessary tasks into a list:

1. define a `FFA` and a `FGA` with their `XfrKeyPair` known to all
2. add some special logics to check transactions related to the `FFA` and `FFA`
3. create a customed `txn_builder` static library for `Go FFI`
4. add necessary codes in the `Tendermint` to support the logics mentioned above