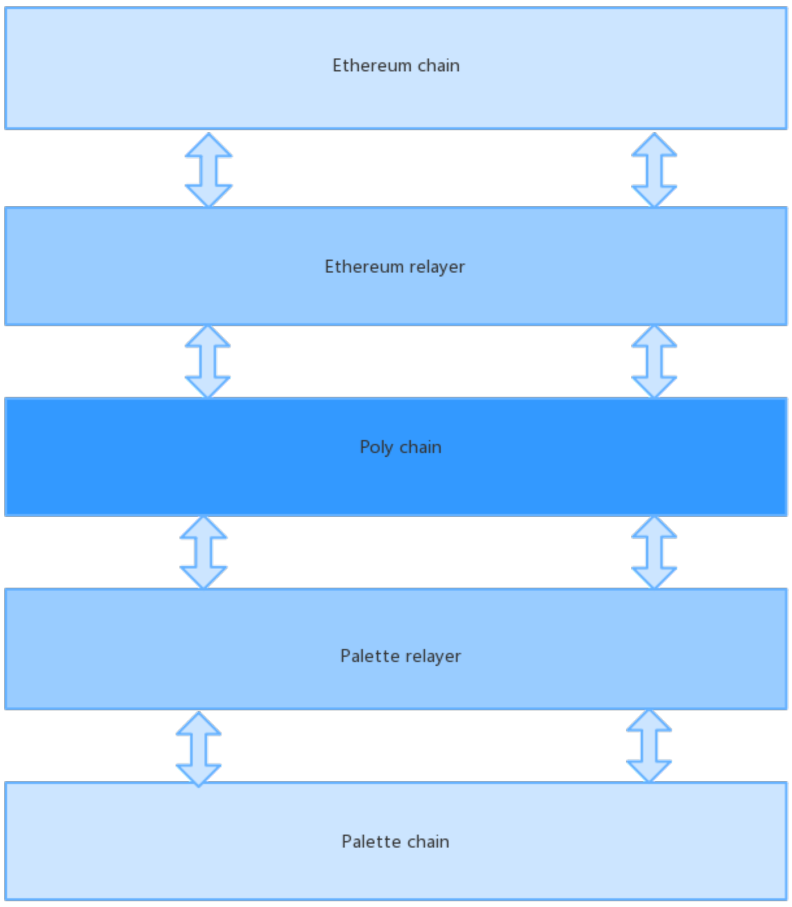
Index

Introduction 2
PRC20 Standard and interfaces 3
    Interfaces 3
PRC721 Standard and interfaces 5
    Interfaces 5
Transaction format 6
Cross-chain protocol and data flow 7
    Framework: Ethereum <-> Palette Chain 7
    PLT Token Cross-Chain Flow: Ethereum -> Palette Chain 8
    PLT Token Cross-Chain Flow: Palette Chain -> Ethereum 9
Ethereum contract and interfaces 10
    Interfaces 10
Palette chain structure 11
Native contract execution 12
Native contract storage 12
Gas fee 13
    Deploy NFT contract 13
    Mint NFT 13
PLT token contract 13
    Extra Interfaces: 14
NFT native contract 14
Governance contract 15
    Interfaces 16
Validator management 17
    Interfaces 18
Economic contract 18
    Basic Reward 20
    Additional Reward 20
    Stake 21
    Interfaces 21

Introduction

This document will describe in detail the overall system architecture and the internal implementation interfaces of each module. In addition, here we will also describe the PRC20 token standard adopted by Palette and the PRC721 standard adopted by NFT token. PRC denotes Palette Chain Request for Comments.

There are five layers that are part of the architecture:
. Ethereum chain, PLT token and Inflation contracts deployed on this chain
. Ethereum relayer, listen events on ethereum chain and push them into poly-chain
. Poly network(poly-chain), listens and exchanges events between the two relayers
. Palette relayer, listen events from palette chain
. Palette chain, manage native contracts for mint PLT, NFT asset, staking, rewarding, gasFee etc.



And functions implemented in the form of modules are as follows:
. Ethereum contracts, contracts on ethereum-chain
. Cross-chain protocol, processes data flow

. Palette native contracts, native contracts for majority of the functions

PRC20 Standard and interfaces

PLT tokens issued on Ethereum and Palette use a token standard called PRC20. This specification can increase the interchangeability of PRC20-based tokens and perform the same operations in dApps.

PRC20 deals with the problem of the Ethereum community creating unique tokens and functions and solves the issue of destroying smart contracts and cyber attacks during token transfers.

The common functionality token (Utility token) is also an application type based on PRC20. PRC20 contains six functions, two events, and three token information functions. As long as it is written in the smart contract, it can be regarded as the token based on PRC20.

We will briefly describe the API methods here.

Interfaces

1.Name: name

Solidity format: function name() public view returns (string)

Description: Returns the name of the token - e.g. "PaletteToken".

Name: symbol

Solidity format: function symbol() public view returns (string)

Description: Returns the name symbol of the token - e.g. “PLT”.

Name: totalSupply

Solidity format: function totalSupply() public view returns (uint256)

Description: Returns the total supply of tokens.

4.Name: decimals

Solidity format: function decimals() public view returns (uint8)

Description: Returns no. of decimal places which the amount of this token can reach. Generally, the set value is 18, which means that the amount can be precise to 18 digits after the decimal point.

5.Name: balance

Solidity format: function balanceOf(address _owner) public view returns (uint256 balance)

Description: Returns the token balance of the `_owner` address.

6.Name: transfer

Solidity format: function transfer(address _to, uint256 _value) public returns (bool success)

Description: transfer `_value` amount of tokens to address `_to`, and must fire the Transfer event.The function should throw if the message caller's account balance does not have enough tokens to spend.

7.Name: transferFrom

Solidity format: function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)

Description: transfer `_value` amount of tokens from address `_from` to address `_to`, and trigger the Transfer event. The transferFrom method is used for a withdrawal workflow, allowing contracts to transfer tokens on your behalf. This can be used, for example, to allow a contract to transfer tokens on your behalf and/or to charge fees in sub-currencies. The function should throw an exception unless the `_from` account has deliberately authorized the sender of the message via some mechanism.

8.Name: approve

Solidity format: function approve(address _spender, uint256 _value) public returns (bool success)

Description: Allows `_spender` to withdraw from your account multiple times, up to the `_value` amount. If this function is called again it overwrites the current allowance with _value.

9.Name: allowance

Solidity format: function allowance(address _owner, address _spender) public view returns (uint256 remaining)

Description: Returns the amount which `_spender` is still allowed to withdraw from `_owner` address.

PRC721 Standard and interfaces

NFT issued on the Palette are issued using the token standard called PRC721 which differs from PRC20 in that it allows for a completely new type of token implementation: non-fungible tokens.

Palette has adopted the PRC721 standard specification for the implementation of NFT, a technological innovation that records fan contributions to content holders and also enables secondary distribution. These are token standards implemented on Ethereum, but they are also reflected in Quorum.

Prior to the PRC721 proposal, it was only possible to handle alternative tokens using the PRC20 standard and other standards. However, with the advent of PRC721, each token can

Interfaces

1.Name: balanceOf

Solidity format: function balanceOf(address _owner) external view returns (uint256)

Description: returns the number of NFTs owned by `_owner`, it possibly zero

2.Name: ownerOf

Solidity format: function ownerOf(uint256 _tokenId) external view returns (address)

Description: Fetch the owner of an NFT token, return the address of the owner of the NFT which is identified by `_tokenId`.

3.Name: safeTransferFrom

Solidity format: function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable

Description: Transfers the ownership of an NFT from one address to another address.

4.transferFrom

Solidity format: function transferFrom(address _from, address _to, uint256 _tokenId) external payable

Description: transfer ownership from `_from` to `_to` address of an NFT which identified by `_tokenId`.

5.Name: approve

Solidity format: function approve(address _approved, uint256 _tokenId) external payable

Description: change or reaffirm the approved address `_approved` for an NFT which identified by `_tokenId`

6.Name: setApprovalForAll

Solidity foramt: function setApprovalForAll(address _operator, bool _approved) external

Description: enable or disable approval for a third party to manage all of `msg.sender``'s assets. `_operator` denotes the address to add to the set of authorized operators, if `_approved` is true it denotes add, while false denotes revoke.

7.Name getApproved

Solidity format: function getApproved(uint256 _tokenId) external view returns (address)

Description: get the approved address for a single NFT which identified by `_tokenId`

8.Name: isApprovedForAll

Solidity format: function isApprovedForAll(address _owner, address _operator) external view returns (bool)

Description: query if the of `_operator` address is an authorized operator for the `_owner` address

Transaction format

Palette user should send transactions by jsonrpc or rest api, and the format of transaction as follow:

```
type Transaction struct {
	from common.Address
```

```
        to common.Address
        nonce uint64
        amount *big.Int
        gasLimit uint64
        gasPrice *big.Int
        data []byte
}
```

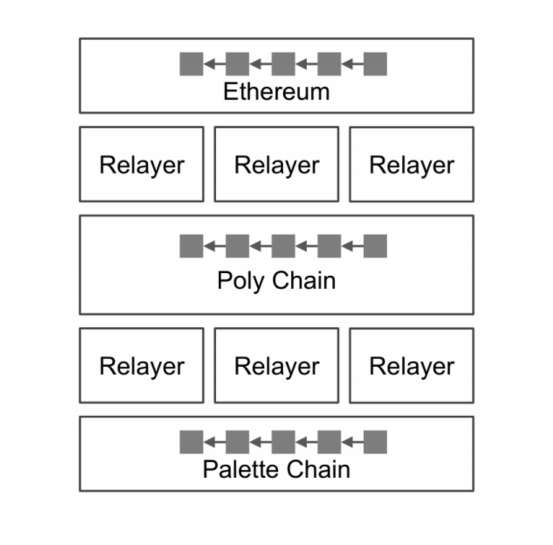
. from: address of the transaction sender
. to: address of the transaction destination. Palette manages a fixed, non-repeated effective address internally, and implements the routing relationship between contract addresses and native contract objects through native contract router. it should be empty bytes array or nil when deploying the NFT assert contract.
. nonce: sequence of the transaction sender account
. data: transaction payload, this field should be empty bytes array or nil while deploying NFT asset contract.
. gasLimit: Generally, a regular Ether (ETH) transaction would be made with, at least, a 21,000 gas limit. If the gas limit and gas price (Gwei) are set high, the operation will occur much faster. And the gas price on the Palette chain is inoperative, the value should be 0.

Cross-chain protocol and data flow

Cross-chain protocol is designed for realizing blockchain interoperability, building the next generation internet. It is based on the side-chain/relay mode and adopts a two-layer architecture. It employs Poly chain as a cross-chain coordinator, multiple homogeneous chains as cross-chain transaction executors, and relayers as a cross-chain information porter.

By addressing things such as trust, security and transaction issues with respect to chain data, Poly chain has implemented a safe, easy-to-use and efficient cross-chain system.

Framework: Ethereum <-> Palette Chain



As illustrated in the above figure, the cross-chain framework consists of the Palette chain, the Palette relayer, Poly chain, the Ethereum relayer, and Ethereum. To put it simply, the user's transaction proof on Ethereum is first transferred to Poly chain by the relayer, and then transferred to a Palette chain by their Palette relayer.

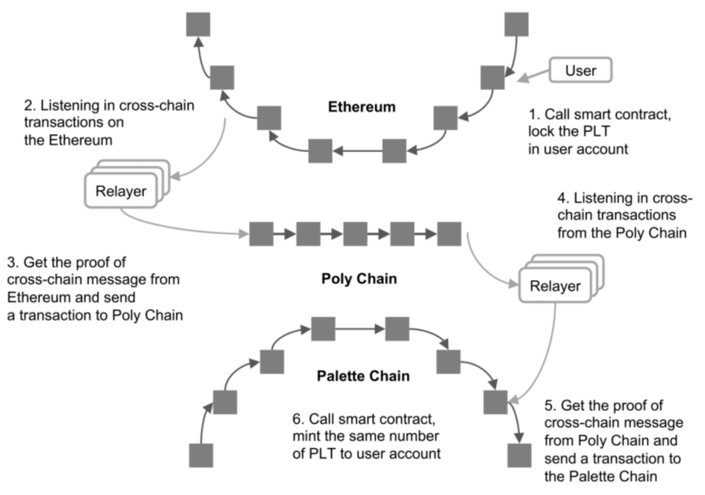
The parties involved in this process, as shown above, are:

. Poly chain: Poly chain is one of the crucial components within this cross-chain ecosystem. Every type of node is deployed and maintained by different individuals or organizations and has its unique governance and trust mechanism. The Poly chain is responsible for establishing a link, verifying the validity of cross-chain messages, maintaining the uniformity of the cross-chain parameters, etc.

. Relayer: Every chain has a relayer linked to them which monitors the transactions taking place in the corresponding networks. They basically transmit the transaction information to the relay chain, thus connecting the relay chain with the outside world.

. Palette Chain: The Palette chain is a consortium blockchain. It supports interoperability with Ethereum using cross-chain technology.

PLT Token Cross-Chain Flow: Ethereum -> Palette Chain


User
1. Call smart contract, lock the PLT in user account
Ethereum
2. Listening in cross-chain transactions on the Ethereum
Relayer
4. Listening in cross-chain transactions from the Poly Chain
3. Get the proof of cross-chain message from Ethereum and send a transaction to Poly Chain
Poly Chain
Relayer
Palette Chain
6. Call smart contract, mint the same number of PLT to user account
5. Get the proof of cross-chain message from Poly Chain and send a transaction to the Palette Chain

The process flow of cross-chain transactions from Ethereum to Palette chain:

1. Users send a cross-chain transaction to Ethereum. It will call a contract to lock some PLT tokens in the user account. If the account balance is sufficient, the user’s PLT will be locked in the contract, and a cross-chain message will be generated that will be packed in the block header of the next block.

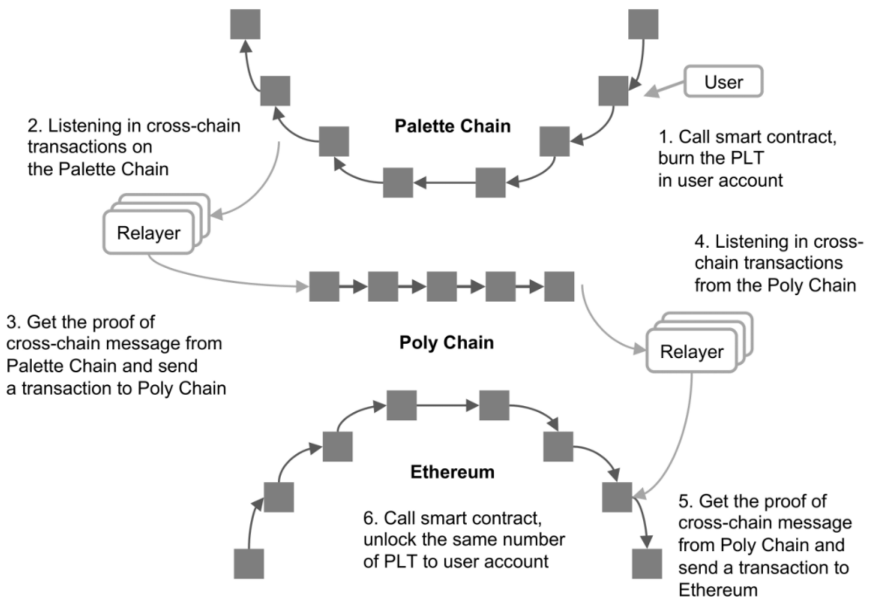
2. The Ethereum relayer is constantly listening for cross-chain transactions from Ethereum. If it receives a cross-chain message, it will fetch the proof of this cross-chain message and send a cross-chain transaction to Poly chain.

3. Poly chain will then verify whether this cross-chain transaction is valid. If the cross-chain message is validated successfully, Poly chain will generate a new cross-chain message.

4. The Palette chain relayer is listening for cross-chain transactions to the Palette chain. If it receives a cross-chain message, it will get the proof of this cross-chain message and send a cross-chain transaction to Palette chain.

5. If Palette chain receives the cross-chain transaction, it will verify whether it is valid. If the cross-chain message is validated successfully, it will call a PLT contract which is the same as PRC20 on Ethereum and mint the same amount of PLT on Palette chain and transfer them to the user wallet.

PLT Token Cross-Chain Flow: Palette Chain -> Ethereum


User
Palette Chain
1. Call smart contract, burn the PLT in user account
2. Listening in cross-chain transactions on the Palette Chain
Relayer
4. Listening in cross-chain transactions from the Poly Chain
3. Get the proof of cross-chain message from Palette Chain and send a transaction to Poly Chain
Poly Chain
Relayer
Ethereum
5. Get the proof of cross-chain message from Poly Chain and send a transaction to Ethereum
6. Call smart contract, unlock the same number of PLT to user account

The process flow of cross-chain transactions from Palette chain back to Ethereum:

1. Users send a cross-chain transaction to the Palette chain. It calls a contract to deduct the PLT from the user account. If it successfully confirms the user has sufficient account balance, the contract will deduct the PLT amount from the user account and a cross-chain message will be generated, which will then be packed in the block header of the next block.

2. The Palette chain relayer is constantly listening for cross-chain transactions from the Palette chain, and if it receives a cross-chain message it fetches the proof for this cross-chain message and sends a cross-chain transaction to Poly chain.

3. Poly chain will verify whether the cross-chain transaction is valid. If the cross-chain message is validated successfully, Poly chain will generate a new cross-chain message.

4. The Ethereum relayer is listening for cross-chain messages from Poly chain. If it receives a cross-chain message, it will get the proof of this cross-chain message and send a cross-chain transaction to Ethereum.

5. If Ethereum receives the cross-chain transaction, it will verify whether it is valid. If the cross-chain message is validated successfully, it will call the PLT contract on Ethereum, and release the same number of PLT and transfer them to the user wallet.

Ethereum contract and interfaces

There are two contracts that will be deployed on the Ethereum chain:

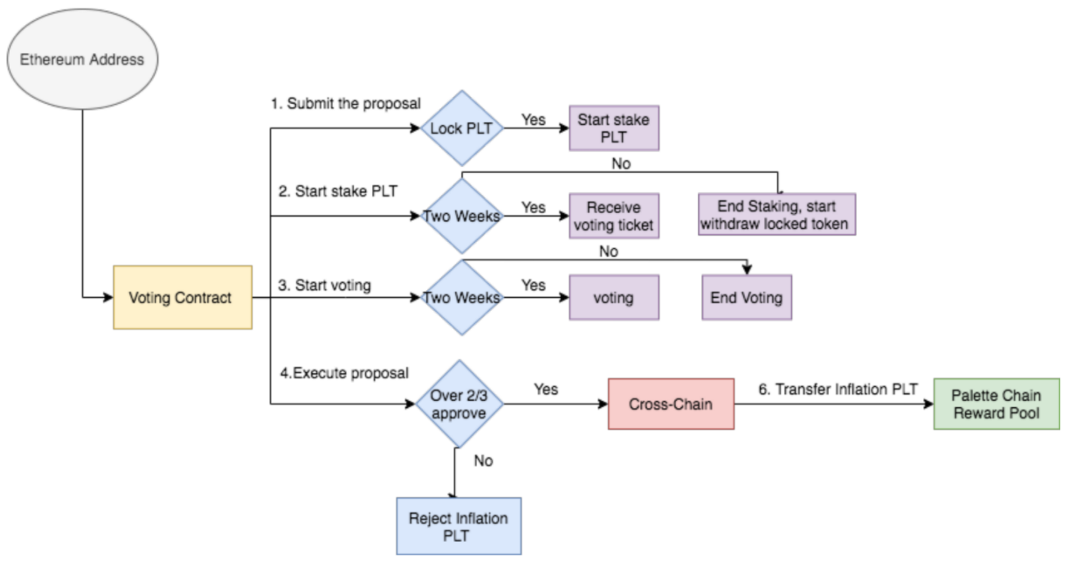
. PLT token contract

. Voting contract

PLT token contract implements the PRC20 standard specification mentioned above.

The Voting contract on Ethereum is in control of the PLT inflation process.

Voting takes place within the voting contract. Users transfer PLT tokens to the voting contract address to carry out staking. Any stakeholder with a PLT token can submit suggestions and participate in the voting process. Each stakeholder is given the right to vote based on their PLT holdings, and the proposal is approved by an approval process that requires at least two-thirds of the valid votes in favour.



Interfaces

1.Name: propose

Solidity format: function propose(address _staker, uint256 _inflation_amount, string desc) public returns (uint256)

Description: Stakeholder `_staker` puts forward an inflation proposal with amount of `_amount` and proposal description, returns the 'proposalId'.

2.Name: stake

Solidity format: function stake(address _staker, uint256 _amount, uint256 _proposalId) public returns (bool success)

Description: Stakeholder `_staker` transfer `_amount` PLT to the voting contract address as locking stake for the proposal which is identified by `_proposalId`.

3.Name: vote

Solidity format: function vote(address _staker, uint256 _amount, uint256 _proposalId) public returns (bool success)

Description: Stakeholder `_staker` votes for the proposal `_proposalId`, the vote `_amount` should be no more than the staking amount.

Palette chain structure

The structure of the Palette chain can be described as follows:

. 3-level Storage Module: The palette storage uses 'stateObject' to record the global state of an account,' 'stateDB' caches data for a short period and 'levelDB' is a persistence module.

. Consensus Module: consensus module implemented by an p2p network with PoA consensus. Block proposal and voting for global params take place in this module.

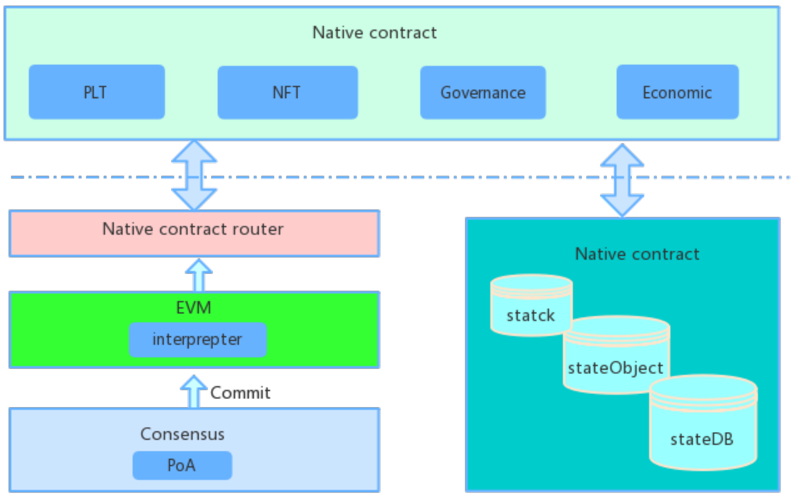
. Virtual Machine: An interpreter is used to handle virtual machine operations set on Ethereum. The native contract module on Palette chain constructs a new interpreter, filters out the virtual machine operations set, and handles native functions directly with golang language.

. Native Contract Router: Routes transactions to the appropriate native contract method with the decoded ABI string.

. A Series of Contracts: Palette native contracts include:

1. PLT token contract
2. NFT asset contract
3. Governance contract
4. Economic contract, used to implement mechanisms such as staking,reward distribution and transaction fees

These four parts are related to chain code. Let us focus on native contracts.



Native contract execution

Palette is developed based on the Quorum chain. The procedure is as follows:

1. Re-construct the virtual machine interpreter to retain the original operations set

2. Develop native contracts in golang

3. Set up a router to redirect the content of the interpreter to the appropriate native contract

4. Add new storage in the native contracts to ensure the correctness of the global state

5. Palette chain undergoes a hard fork when native contracts are updated

Native contract storage

In Quorum, the storage process of ‘eventLog’ and ‘transfer’ are integrated into the virtual machine. Palette needs to include the storage process in the native contract processing and ensure it’s security and validity.

A hash function is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.

At present, the storage of contract accounts only supports the map from hash to hash, e.g:

NFT token info storage:
1. hash("Name") -> Token Name
2. hash("Supply") -> Token Supply
3. hash("Symbol") -> Token Symbol
4. hash("Admin") -> Token Admin

NFT token balance storage：
1. hash(Account address) -> balance
2. hash(Approve address, Account address) -> balance

Gas fee

In the case of public blockchains, gas fees can be set to protect the network from DDos attacks and malicious miners, while Palette chain does not need to be designed for DDos attacks. To improve the user experience, users can make transactions on the Palette chain for free. This means that users who submit certain transactions on Palette chain do not have to pay gas fees. However, as an incentive for consensus nodes, we set gas fees in some of the operations described below, and the fees are distributed to the Palette Consortium through the Palette Reward Pool.

Gas fee will be charged on Palette chain in the following cases:

Deploy NFT contract

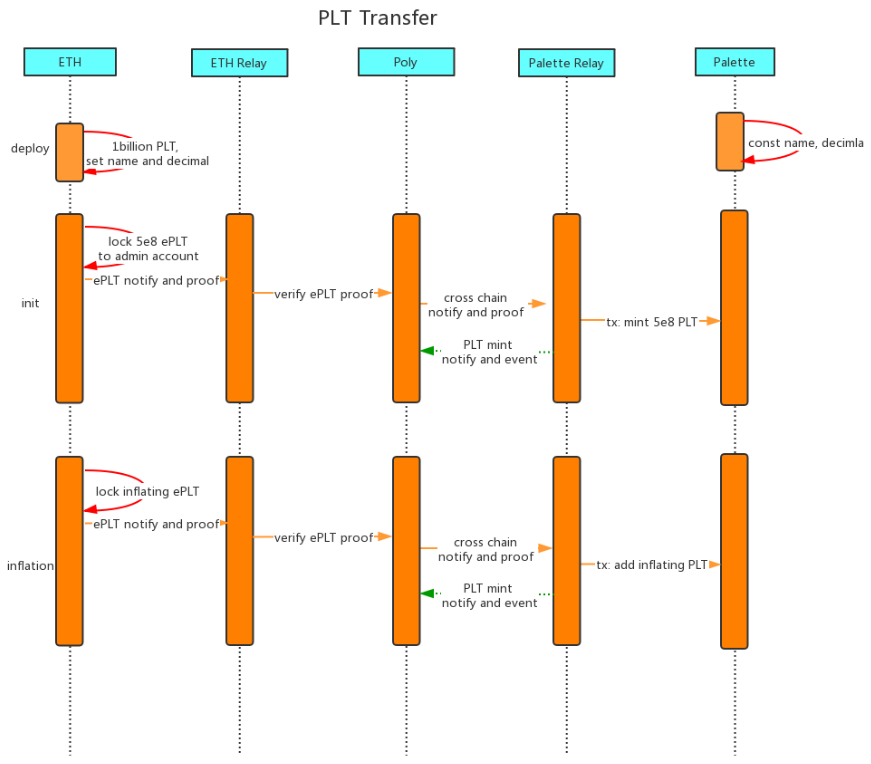
Any NFT project can deploy NFT smart contracts on the Palette chain. Gas fee is required to deploy contracts on Palette, and is calculated in PLT and needs to be paid by the party deploying the contract.

Mint NFT

If the NFT smart contract is successfully deployed, the NFT project can mint NFT on Palette Chain. The gas fee required to issue the NFT will be paid in PLT. In addition, the PLT price is does not remain constant, so the fee structure will be adjusted by the voting system described below.

PLT token contract

The PLT contract mints PLT token on the palette chain and has also adopted PRC20 standard specification. There will be a total amount of 340,000,000 PLT minted in this contract, which will be released over a period of 6 years. The figure illustrates how to initialize this contract and inflate the PLT when necessary. \`ePLT\` in the figure denotes PLT token issued on Ethereum, \`PLT\` denotes mint token in palette chain.



Extra Interfaces:

1.Name: initialize

Golang format: func initialize(value *big.Int) *big.Int

Description: mint the amount of PLT in palette chain. return current total amount.

2.Name: inflation

Golang format: func Inflation(value *big.Int) (*big.Int, error)

Descripton: Inflate PLT in palette chain, return current total amount. Return error if inflation failed.

NFT native contract

NFT tokens are used to deploy and manage NFT assets. The NFT contract implements interfaces based on the PRC721 standard specification. It also have a `deploy` method which can be used to issue different kinds of NFT assets using only one template:

Interfaces

1.Name: deploy
Golang format: func deploy(
name string,
totalSupply *big.Int,
decimal uint64,
owner common.Address,
) error

Description: Issue NFT asset with `name` and a total amount of `totalSupply`. `owner` specifies the ownership address of the asset, and `decimals` sets the precision to which the amount of this token can be specified. Generally, the set value is 1, which means that it can reach 1 digits after the decimal point. Returns an error if the issuing process fails.

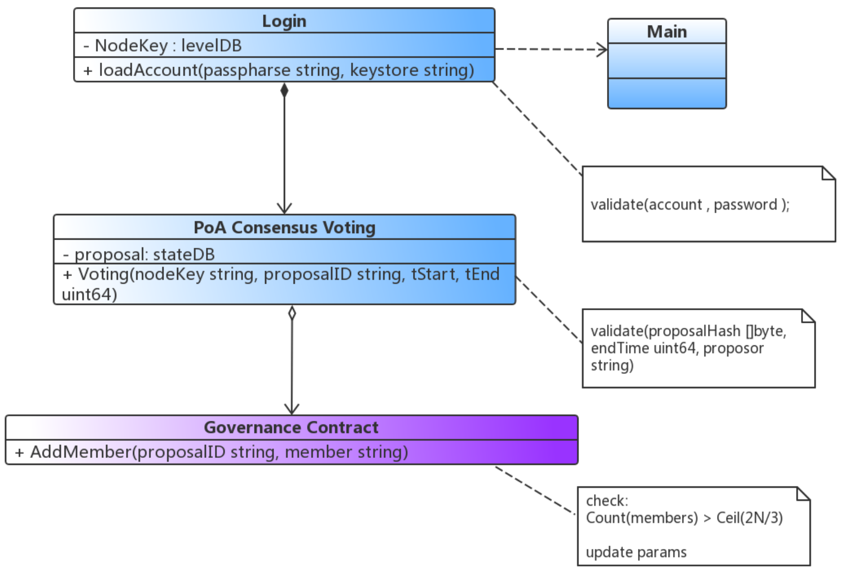
2.Name: info
Golang format: func info() (
name string,
totalSupply *big.Int,
decimal uint64,
owner common.Address,
err error,
)

Description: Query asset related information. Returns error if asset does not exist.

Governance contract

Governance contract records a group of global parameters which have default values. Any validator can propose to update the global parameters, and for this the validator sends a transaction to the governance contract. If the proposal operation succeeds, the contract will record the number of validators and then if the majority agree with the proposal, the contract will update the parameter.

Note: The majority number of validators should be more than Ceil(2N / 3).


Login
- NodeKey : levelDB
+ loadAccount(passpharse string, keystore string)
Main
validate(account , password);
PoA Consensus Voting
- proposal: stateDB
+ Voting(nodeKey string, proposalID string, tStart, tEnd uint64)
validate(proposalHash []byte, endTime uint64, proposor string)
Governance Contract
+ AddMember(proposalID string, member string)
check:
Count(members) > Ceil(2N/3)
update params

Global parameters including:

. GasFee	// global gas fee for deploy NFT contract
. BaseRewardRate	// global base reward rate for block proposal
. RewardPeriod	// global reward time interval
. MintPrice	// global price for mint NFT

Interfaces

1.Name: proposeGasFee

Golang format: func proposeGasFee(validator common.Address, fee *big.Int) error

Description: Validator proposes to update gas fee. Returns error if proposal fails.

2.Name: getGasFee

func getGasFee() (*big.Int)

Description: Get current gas fee. Returns 0 if gas fee value does not exist.

3.Name: proposeBaseRewardRate

Golang format: func proposeBaseRewardRate(validator common.Address, baseRate *big.Int)

Description: Validator issues a proposal to update the reward ‘baseRate’. ‘baseRate’ is converted from a float number which is precise to two decimal places, eg. 32.64% -> big.NewInt(3264) baseRate + extraRate = big.NewInt(10000)

4.Name: getRewardRate

Golang format: func getRewardRate() (baseRate, extraRate *big.Int, err error)

Description: Returns the reward rates, returns error if no rates are recorded in the contract.

5.Name: proposeRewardPeriod

Golang format: func proposeRewardPeriod(validator common.Address, period uint64) error

Description: Validator proposes the new reward period. Returns error if the proposal fails. `validator` denotes the node address, and `period` is a UNIX timestamp, unit being seconds.

6.Name: getRewardPeriod

Golang format: func getRewardPeriod() (uint64, error)

Description: Query the interval time of issuing rewards. Returns error if value does not exist.

7.Name: proposeMintPrice

Golang format: func proposeMintPrice(price *big.Int) error

Description: Validator proposes the price to mint NFT tokens. Returns error if proposal fails.

8.Name: getMintPrice

Golang format: func getMintPrice() (*big.Int, error)

Description: Query the price for minting NFT tokens.

Validator management

After reaching a consensus offline, Palette uses the administrator account in the genesis block to add/remove validators. Administrator operations will be sent to the consensus node network in the form of proposals. Unlike block proposals, nodes only need to use the administrator's public key to verify the signature of the add/delete action, and directly store the node's 'nodeKey' in the database after passing.

Interfaces

1.Name: addValidator

Golang format: func addValidator(
validator common.Address,
salt string,
adminPublicKey *ecdsa.PublicKey,
signature []byte,
delete bool,
) (bool, error)

Description: The admin adds node with address `validator` as validator, this admin account should provide a variable proof `signature` and `adminPublicKey` for other validators for verification. `salt` is an extra field used by admin to ensure the security of the signature. `delete` denotes whether to delete the validator. Returns false and an error if the addition process failed.

2.Name: checkValidator

Golang format: func checkValidator(validator common.Address) (bool, error)

Description: check if the validator is available. return (false, error) if validator not exists.

3.Name: getValidators

Golang format: func getValidators() []common.Address

Description: returns the string slice of validators' addresses. return empty string slice if non validators.

Economic contract

This contract manages the process of recording the gas fee, stake info record and rewards in the following manner:

1.System will deduct part of the user's PLT tokens to this contract address as gas fee when issuing an NFT asset. The contract will record the gas usage and the issuer.

2.Staked PLT assets will be minted and recorded in this contract.

3.Reward info for every block proposal will also be recorded in this contract. The contract address is used as the reward pool.

The total number of PLT to be issued in the form of block rewards is 340,000,000. The total amount of block rewards will be released over a period of 6 years. A certain amount of PLT is unlocked based on the block generation on the Palette chain and is deposited in the reward pool each time. Initially, 67% of the total amount of block reward will be released in the first 3 years after launch. After that, the amount to be released per block will be halved, and 33% of the total block reward will be released in the next three years.

$$R_{b,y} = R_t * P_r * B_s / 3$$

Rb,y: reward for each block

Rt: total amount of block reward(340,000,000)

Bs: block generation speed 1/(86400 * 365 / 10)

Pr: reward factor, 67 % for the first three years, and 33% in the next three years

Upon calculating, it can be determined that for the first three years, the amount that will be released in the form of rewards is 24 PLT, and for the following three years is 12 PLT. The critical block height is 9460800.

PLT usage	percent	amount
inverstor	27%	270,000,000
block reward	34%	340,000,000
team	27%	270,000,000
partner	12%	120,000,000

year	total reward amount	reward/perBlock
1st	75933333	24
2nd	75933333	24
3rd	75933333	24
4th	37400000	12
5th	37400000	12
6th	37400000	12

change block height	block generating speed(second)
9460800	10s

There are two reward factors on the Palette chain in the form of which each member receives a fee to operate a node.

Basic Reward

The basic reward is distributed according to the number of nodes operated by each member. In principle, each party is expected to operate one node at a time. However, if more nodes are needed to improve decentralization or achieve stability, the Palette consortium can make adjustments.

The basic reward `Rb, i` for each member of the Palette consortium is calculated using the following formula.

$$R_{b,\,i} = \alpha P\ (N_i / N_{all})$$

Rb, i : Basic reward for a member
α : Basic reward factor to determine distribution ratio of basic reward
P : Total number of PLT held by Palette Reward Pool
Ni : Number of nodes operated by the members
Nall : Total number of nodes in Palette Chain

At the time of the launch of Palette Chain, `$\alpha$ = 0.2` is the default setting, and changes will be proposed and approved by voting by Palette Consortium members.

Additional Reward

Additional reward will be distributed according to the amount of PLT locked by each member. The additional reward `Ra,i`for each member of Palette Consortium is calculated by the following formula.

$$R_{a,\,i} = (1-\alpha)\ P\ (L_i / L_{all})$$

Ra, i : Additional reward for a member
α : Basic reward factor to determine distribution ratio of basic reward
P : Total number of PLT held by Palette Reward Pool
Li : The total number of PLT locked up for the validator
Lall : The total number of PLT locked up for all nodes.

Here, the total number of PLT locked by the member, Li, includes the delegation from users, allowing companies to attract delegation from users and increase additional reward by demonstrating the robustness and transparency of their node operations and their contribution to revitalizing the ecosystem.

That is, the total number of PLT locked by each company, Li, is shown as follows

$$Li = li + Di$$

li : Total number of PLT locked up by the member themselves

Di : Total number of PLT delegated by fans and investors

That is, the node operation fee Ri which each member receives can be expressed as follows:

$Ri = Rb, i + Ra, i = \alpha P\ (Ni / Nall) + (1-\alpha)\ P\ (li + Di / Lall)$

Stake

Fans and investors in Palette ecosystem can also participate in the governance of Palette Chain by staking their PLT to specific members of Palette Consortium. Stake, here, means that PLT you hold is tied to specific members and locked on a smart contract. This means that members who are staked more will receive more additional reward.

Members can also distribute PLT as staking rewards to PLT holders who staked. Each member can set a percentage to distribute PLT received as node operation fee. This percentage can be set by each member at any time between 0% and 100%. The staking reward Rd, j from the member to fans and investors who staked PLT is calculated by the following formula.

$$Rd, j = \beta i\ (Rb, i + Ra, i)\ (Dj, i / Dall, i)$$

Rd, i : Delegation reward for fans and investors

~~βi~~ : Delegation reward factor which is determined by each member

Rb, i : Basic reward of a member

Ra, i : Additional reward of a member

Dj, i : Total number of PLT delegated by users to eligible members

$Dall, i$: Total number of PLT delegated to eligible members

Thus, the delegation by PLT holders corresponds to a voting system to determine node operation fees distributed to each member from Palette Reward Pool in Palette ecosystem. Governance by PLT holders increases the transparency and robustness of node operations by members. Members participating in Palette Consortium will be able to receive more reward through delegation.

Interfaces

1.Name: lastestRewardBlock

Golang format: func lastestRewardBlock() uint64

Description: Query the block number of the latest reward. The economic contract records the rewards of the reward holder for every block, but the actual distribution of rewards takes place in intervals. Returns 0 if rewards were never distributed before.

2.Name: avaliableReward

Golang format: func avaliableReward(owner common.Address) (*big.Int, error)

Description: Returns the sum of the reward for a particular reward holder `owner`

3.Name: batchReward

Golang format: func BatchReward() error

Description: Batch transfers PLT rewards from the economic contract address to the respective rewardHolder addresses.

4.Name: stake

Golang format: func stake(stakeholder common.Address, amount*big.Int) error

Description: Records stakeholder’s staking amount.

5.Name: gas

Golang format: func gas(owner, nft common.Address, amount *big.Int) error

Description: Records gas fee for the NFT asset issued by the owner.