TECHNICAL DOCUMENTATION

Project Overview

NXTreme, the node based on Hyperledger Best client, hosts NXTChain, a public permission network. Such a network has rules dictating what nodes are allowed to connect. The Permissioning smart contracts are an on-chain mechanism for managing and synchronizing these rules among clients.

1. Functional Requirements

1.1. Roles

The relevant roles are as follows:

- **Nodes** – Only authorized nodes may connect to the network.
- **Administrators** – These are on-chain accounts that have the ability to change the rules contracts and to grant and revoke administrative privileges to others.

1.2. Features and Use Cases

There is one contract included as part of the network's genesis block:

- NodeIngress is used to check whether incoming node connections should be allowed. It calls connectionAllowed on the NodeRules contract.

These contracts are a layer of indirection that can be used for upgradeability. Rather than directly implement permissioning rules, they delegate to other contracts, which can be swapped at runtime by administrators.

The following contracts provide the actual rules implementations:

- NodeRules keeps a whitelist of nodes that are allowed to connect to the network.

Finally, the Admin contract is used to keep a whitelist of administrator accounts. These accounts are allowed to add or remove things from the whitelists. They can also swap out the rules contracts altogether.

2. Technical Requirements

2.2. Contract Information

This section contains detailed information (their purpose, assets, functions, and events) about the contracts used in the project.

2.2.1. Admin.sol, AdminList.sol, AdminProxy.sol

Those 3 contracts manage the ingress of new admin, deleting of existing admin, and storing the list of the active admin.

2.2.1.1. Assets

Admin.sol contains:

- **allowlist**: a public array that stores all the admin's address
- **indexOf**: mapping address to 1 or 0 based if they are in the allowlist array

2.2.1.2. Modifiers

Admin.sol has the following modifiers:

- **onlyAdmin()**: check if the msg.sender is an present in the allowlist
- **notSelf(address address)**: check that a method is not invoked with msg.sender as parameter.

2.2.1.3. Functions

Adminlist.sol has the following functions:

- **isAuthorized(address _address):** call the function 'exists(address _account)' from AdminList.sol to check if an address is an admin.
- **addAdmin(address _address)**: allow an admin to call the function 'add(address _account)' from AdminList.sol to add a new admin.
- **removeAdmin(address _address):** allow an admin to call the function 'remove(address _account)' from AdminList.sol to remove an admin.
- **getAdmins():** return the array allowlist.
- **addAdmins(address[] calldata accounts)**: allow an admin to call 'addAll(address[] memory accounts)' from AdminList.sol to add multiple new admins.

AdminList.sol has the following functions:

- **size():** return the number of admins.
- **exists(address _account):** return a bool based on whether _account is an admin.
- **add(address _account):** add a new address to the admin list.
- **addAll(address[] memory accounts):** add multiple addresses to the admin list.
- **remove(address _account)**: remove an address from the admin list

2.2.2. Ingress.sol, NodeIngress.sol and NodeStorage.sol

Those 3 contracts manage the ingress of new admin, deleting of existing admin, and storing the list of the active admin.

2.2.2.1. Assets

Ingress.sol contains:

- **contractKeys**: a public array that stores all the contract keys.
- **indexOf**: mapping contract keys to 1 or 0 based if they are saved in the array or not.
- **registry:** mapping the name of the smart contract with his contract keys.
- **onlyUseEnodeId :** bool variable to use only the enode

NodeIngress.sol contains:

- **version:** a private constant that refers to the version of the smart contract.

NodeStorage.sol contains:
A struct:
Enode: this object contains information about the nodes

- String endeId - the enode of the node
- String ip - the ip of the node
- Uint16 port - the port of the node

And contains:

- **allowlist:** public array of the enode of the nodes
- **indexOf**: maps a uint assigned to a node with 0 or 1 based if the node is in the allowlist
- **latestVersion:** variable that saves an address as latestVersion
- **Owner**: variable that saves an address as the owner

2.2.2.3. Modifiers

NodeStorage.sol contains:

- **onlyLatestVersion():** allow only if the msg.sender is the address saved as 'latestVersion'.
- **onlyAdmin():** allow only if the msg.sender is an Admin or the owner.

2.2.2.3. Functions

Ingress.sol has the following functions:

- **getContractAddress(bytes32 name):** set the name of the contract in 'registry' mapping.
- **getSize()**: returns the number of contract keys saved.

- **isAuthorized(address account):** check if msg.sender is the admin contract or an admin address.
- **setContractAddress(bytes32 name, address addr):** populate the 'registry' mapping.
- **removeContract(bytes32 _name):** remove the info related to a contract from the 'registry' mapping.
- **getAllContractKeys():** returns the contractKeys array.

NodeIngress.sol has the following functions:
- **getContractVersion():** return the 'version' constant.
- **emitRulesChangeEvent(bool addsRestrictions)**: allow only the rules smart contract to change the rules and emit an event.
- **removeAdmin(address _address):** allow an admin to call the function 'remove(address _account)' from AdminList.sol to remove an admin.
- **connectionAllowed(string calldata enodeId, string calldata enodeHost, uint16 enodePort):** check if a determinate node can connect to the network.

NodeStorage.sol has the following functions:
- **upgradeVersion(address _newVersion):** allow only an admin or the owner to change the 'latestVersion' address.
- **size():** return the full list of enode stored in the array 'allowlist'.
- **exists(string memory _enodeId, string memory _host, uint16 _port):** checks if a node is in the enode struct.
- **add(string memory _enodeId, string memory _host, uint16 _port):** allow only latestVersion address to add a new enode in the struct and array.
- **remove(string memory _enodeId, string memory _host, uint16 _port):** allow only latestVersion address to remove an enode.
- **getByIndex(uint index):** retrieve from indexOf mapping the info on the enode struct of a node.
- **setValidateEnodeIdOnly(bool _onlyUseEnodeId):** allow only the latestVersion address to switch the logic to the enode id mapping.
- c**alculateKey(string memory _enodeId, string memory _host, uint16 _port):** convert the combination of the 3 data of the enode struct into the enode id.

2.2.3. NodeRules.sol, NodeRulesList.sol and NodeRulesProxy.sol

Those 3 contracts manage the rules of permission of the nodes.

2.2.3.1. Assets

NodeRules.sol contains:
- **readOnlyMode**: a public bool that enables the read-only mode where rules can't be added or removed
- **version**: a private constant that refers to the version of the smart contract.

NodeRulesList.sol contains:
A struct:
Enode: this object contains information about the nodes
  - String endeId - the enode of the node
  - String host - the ip of the node
  - Uint16 port - the port of the node

2.2.3.2. Modifiers

NodeRules.sol has the following modifiers:
- **onlyOnEditMode()**: allow the function only with read-only mode off
- **onlyAdmin():** allow only if the msg.sender is an Admin.

2.2.3.3. Functions

NodeRules.sol has the following functions:
- **getContractVersion():** return the 'version' constant.

- **isReadOnly**: returns the bool ‘readOnlyMode’
- **enterReadOnly()**: allow an Admin to enable the read-only mode
- **exitReadOnly():** allow an Admin to disable the read-only mode
- **connectionAllowed(string calldata enodeId, string calldata enodeHost, uint16 enodePort):** returns a bool based if the node is allowed to connect to the network
- **enodePermitted(string memory enodeId, string memory host, uint16 port):** returns a bool based if the node is allowed to connect to the network.
- **addEnode(string call data enodeId, string call data host, uint16 port):** allow an admin when the read-only mode is off to add a new enode into the ‘enode’ struct
- **removeEnode(string call data enodeId, string call data host, uint16 port):** allow an admin when the read-only mode is off to remove a enode from the ‘enode’ struct
- **getSize():** return the number of enode stored in the NodeStorage.sol struct
- **triggerRulesChangeEvent(bool addsRestrictions):** emit an even from NodeRulesList.sol to signal a change of the rules
- **setValidateEnodeIdOnly(bool _onlyUseEnodeId):** allow an Admin when the read-only mode is off to switch the logic to the enode id mapping.

NodeRulesList.sol has the following functions:
- **setStorage(NodeStorage _storage):** set the address of NodeStorage.sol
- **upgradeVersion(address _newVersion):**internal the upgrade function from NodeStorage.sol.
- **size():** return the number of enode stored on NodeStorage.sol
- **exists(string memory _enodeId, string memory _host, uint16 _port):** checks if a node is in the enode struct from NodeStorage.sol.
- **add(string memory _enodeId, string memory _host, uint16 _port):** internal function to add a new enode in the struct and array of NodeStorage.sol.
- **remove(string memory _enodeId, string memory _host, uint16 _port):** internal function to remove an enode in the struct and array of NodeStorage.sol.
- **calculateKey(string memory _enodeId, string memory _host, uint16 _port):** convert the combination of the 3 data of the enode struct into the enode id using the function from NodeStorage.sol.
- **getByIndex(uint index):** retrieve from indexOf mapping the info on the enode struct of a node from NodeStorage.sol.
- **setValidateEnodeIdOnly(bool _onlyUseEnodeId):** internal function to switch the logic to the enode id mapping.