NEO Name Service Whitepaper (draft)

Jan. 19th, 2018

01

NNS background

1.1 What is NNS

NNS is the NEO Name Service, a distributed, open, and extensible naming system based on the NEO blockchain. Our primary goal is to replace irregular string such as wallet address and smart contract Hash which are hard to memorize for humans with words and phrases. We will offer ending in".neo" name service first.

Through name service, people don't need to remember address and Hash they don't understand anymore. You could make a transfer or use contracts by just knowing a word or phrase.

NNS can be used to resolve a wide variety of resources. The initial standard for NNS defines resolution for NEO addresses or Smart contracts(ScriptHash), but the system is extensible, allowing more resource types to be resolved in future without NNS upgrades.

1.2 Why do we need NNS?

When Satoshi Nakamoto designed Bitcoin address, he created base58 encode by himself rather than adopting base 64 encode commonly used in coding community. In the base58 encode, he deleted some ambiguous characters: 0(zero), O(capital letter o), I(capital letter i) and l(lower case letter L). This reflects Satoshi Nakamoto's consideration for the usability of blockchain address. However, blockchain address is still not human-friendly enough, because it's too long, hard to memorize and not easy to compare whether it's right or wrong. As blockchain's popularity increases, the shortcomings of its address will be more obvious. As we won't use a 32-byte string as an E-mail account today, alias service could provide a huge help for the usability of blockchain system. As IPFS has its alias service IPNS, and Ethereum ENS, We argue that NEO system should have its own alias service. We call it as NEO Name Service(NNS), NEL community will increase the usability of NEO blockchain by providing NNS service.

The primary usage scenario of alias service is transfer of tokens via alias, especially for those accounts who need to make public their wallet addresses and do not need to change their addresses frequently. For example, when an ICO is underway, the project initiator need to make public its official wallet address on its official website. But even the official wallet address is modified by hackers, it is difficult for investors to notice that. So if the project initiator could make public a short and easy-to-remember address alias, then even it's modified, it could be easily found, thus preventing wallet address from being modified by hackers.

02

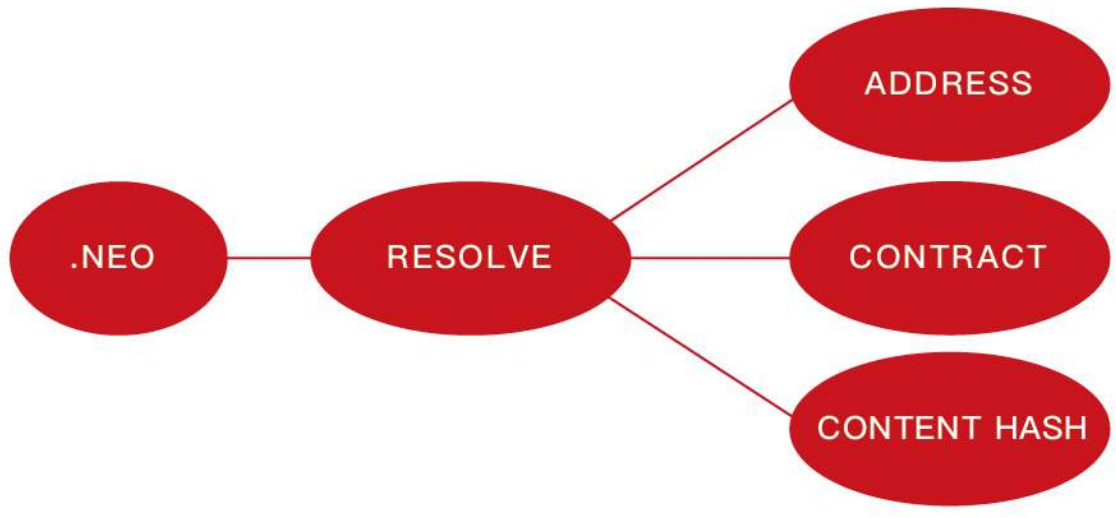
NNS system design overview

What types of resource an alias points to is extensible, as long as corresponding resolver is achieved. Besides pointing to an account address, an alias could also point to a contract address, so smart contracts can be invoked via alias. There will probably be many smart contract templates, thus mistakes could be avoided if the alias service is used to invoke smart contract templates.

As blockchain is the infrastructure of next generation Internet, an increasing number of services will be based on blockchain. For example, the decentralized cloud storage service. File addressing is done through the file's hash value-the only identifier. We could give a hash value an alias such as a file name that could be understood easily, then we map the alias to the file's hash value to achieve the file addressing. So alias service could be used together with NEOFS-the decentralized file storage based on NEO in future. NNS could also provide alias service for decentralized messaging, decentralized email service and so forth as more and more services are being built on NEO.

1.3 The relationship between NNS and ENS

NNS and ENS share the same goal of increasing the usability of blockchain. But they are based on different blockchain platforms and serve different blockchain platforms. We would like to extend our thanks to ENS, because we drew on experience from ENS when we designed NNS system, meanwhile we also made lots of innovative designs. For example, we split the owner contract from registry module to achieve more flexible ownership control. We have two types of resolutions: quick resolution and complete resolution. We have introduced a smart token in our economic model to achieve redistribution of system costs.



2.1 NNS system functions

NNS system has two functions: first, to resolve human-readable name like beautiful.neo into machine-readable identifiers like NEO's address; second, to provide descriptive data for domain names, such as whois, contract interface description and so forth.
NNS has similar goals to DNS. But based on blockchain architecture design, NNS is decentralized and serves blockchain network. NNS operates on a system of dot-separated hierarchical names called domains, with the owner of a domain having full control over the distribution of subdomains.

Top-level domains, like '.neo' and '.gas' are owned by smart contracts called registrars. One registrar manages one root domain name and specifies rules governing the allocation of their subdomains. Anyone may, by following the rules imposed by these registrar contracts, obtain ownership of a second-level domain for their own use.

2.2 NNS Architecture

NNS has four components:

1. Top-level domain name contract.(domain name root is the script that manages root domain name.)
2. The owner (the owner could be a personal account's address or a smart contract)
3. Registrar (A registrar is simply a smart contract that issues subdomains of that domain to users. The root domain name specifies a root domain name's registry.)
4. Resolver (is responsible for resolving a domain name or its subdomain names.)

2.2.1 Top-level Domain Name Contract

Domain name root is the manager of all the information of a root domain name such as .test. No matter it is a second-level domain name like aa.test or third-level domain name like bbb.aa.test, both of their owners are kept in the domain name root which keeps the following data in the form of dictionary.

1. The owner of the domain name
2. The registrar of the domain name
3. The resolver of the domain name
4. The TTL(time to live) of the domain name

2.2.2 Owner

The owner of the domain name could be either an account address or a smart contract. (ENS's design is a smart contract that owns a domain name is called registrar. Actually registrar is the owner's exception. We separate the owner of the domain name from the registrar, making the system clearer.)

Owners of domain names may

1. Transfer the ownership of the domain name to another address.
2. Change registrar. The most common domain registrar is " administrators allocate subdomains manually."
3. Change the resolver.

A smart contract is allowed to be the owner, which provides a variety of ownership models.

1. The domain name co-owned by two persons. The transfer of domain names or changing registrars is only possible with two people's signatures.
2. The domain co-owned by multi-person. The transfer of domain names or changing registrars is only possible with more than 50% of people's signatures.

If the owner of the domain name is an account address, the user could invoke registrar's interface to manage second-level domain names.

2.2.3 Registrar

(ENS's design is a smart contract which owns top-level domains is called registrar. Actually the registrar is the owner's special case. Our separation of the owner of the domain name from the registrar makes the system clearer. Most users don't sell their second-level domain names, so most users just need to configure a resolver rather than a registrar.)

A registrar is responsible for specifying subdomain names of a domain name to other owners. The registrar operates by invoking the domain root's script. The domain name root checks whether the registrar has the authority to operate this domain name.

The registrar has two functions.

1. To re-specify subdomain names of a domain name to other owners.
2. To check whether the owner of a subdomain name is legal or not, because second-level domain names could be transferred to others after third-level domain names are

3. then bbb.aaa.neo corresponding
4. hashB = hash256(hashA+"bbb")
5. then ccc.bbb.aaa.neo corresponding
6. HashC = hash256(hashB+"ccc")

This definition allows us to store all levels of domain names, level 1, level 2, to countless levels, in a data structure: Map <hash256, parser> in a flat way.

This is exactly how the registrar saves the resolution of domain names.

This recursive calculation of NameHash can be expressed as a function: Hash = Name-Hash ("xxx.xxx.xxx ..."); for the realization of NameHash, please refer to Section 3.2.

2.2.5.2 Resolution process

The user invokes the resolution function of the root domain name for resolution, and the root domain name provides both complete and quick resolution. You can invoke it as need. You can also query the resolver and invoke it by yourself.

Quick resolution

Quick resolution root domain name directly searches the resolver of a complete domain name. if not, search the parent domain name's resolver and then invokes the resolver for resolution.

There are fewer operations for quick resolution, but there's a flaw: the third-level domain name is sold to someone else and the resolver exists, but the second-level domain name has been transferred. At this point the domain name can still be resolved.

Complete resolution

In the complete manner, the root of the domain name will start with the root domain name and queries ownership and TTL layer by layer. It will fail if they don't comply with.

More operations are needed in the complete resolution and operations has a linear growth with the layer number of domain names.

2.3 Economic Model-lock-free, cyclically redistributed NNC token

sold . In the complete resolution, the registrar will be asked whether its subdomain names are allocated to specified owners. If not, the resolution is invalid.

The registrar is a smart contract. There could be many types of registrars.

1. “First come, first served” registrar. Users are free to grab domain names. Our test net’s .test domain names will be registered in a “first come, first served” way.
2. Administers allocate registrars manually. An administrator sets up how the ownership of subdomain names is processed. Individuals with second-level domains allocate subdomain names manually.
3. Registrar auction. Testnet’s .neo domains and mainnet’s .neo domain names will be registered in the way of auction with collaterals.

2.2.4 Resolver

NNS’s main function is to finish the mapping from the domain name to the resolver. The resolver is a smart contract which interprets names into addresses.

Any smart contracts which follow NNS resolver rules could be configured to resolvers. NNS will provide general-purpose resolvers.

If new protocol types needs to be added, direct configuration can be done without changing NNS system if current NNS rules are not disrupted.

2.2.5 Resolution rules

2.2.5.1 Domain name storage

The domain name NNS stores is 32 byte hashes, rather than plain text of the domain name. Here are reasons for this design.

1. A unified process allows any length of the domain name.
2. It preserves the privacy of the domain name to some extent.
3. The algorithm for converting a domain name to a hash is called NameHash, and we'll explain it in other documentation. The definition of NameHash is recursive.

```
1.for example aaa.neo corresponding
2.hashA  =  hash256(hash256(“.neo”) + “aaa”)
```

NNS system will issue a built-in token called NNC.

NNC has three functions:

a) NNC can be used as the collateral assets in the auction. .neo domain names will be handed out via an auction process. In the bidding, who bids the most NNC wins the ownership of the domain name. NNC used in auction will be locked temporarily. The ownership of locked NNC belongs to the owner of domain names, which means that the ownership of NNC will be transferred as the ownership of domain names is transferred.

b) NNC can be used to pay domain name rent. Because NNC is just locked in the auction, causing no other losses except losing NNC liquidity, so in order to prevent speculators from maliciously bidding to drive up domain name prices, it's needed to introduce rent mechanism for price adjustments: every year (or other fixed time) domain names are charged a certain rent as the domain name use cost. We will first open up second-level domain names of more than 5 characters without charging rent. We will consider introducing a rent mechanism when less-than-5 character high value domain names are fully open.

c) System income redistribution. During the bidding process, the system will charge fees to prevent malicious bidding. Besides that the system will also have rental revenue. these revenues will eventually be returned to NNC holders in proportion to their NNC's holdings. In order to facilitate the redistribution of system revenue, we added the concept of coin days for NEP5 tokens, and NNC token holders only need to manually collect a bonus at intervals. lock-free cyclical redistribution of NNC tokens is achieved in this way.

Issuance volume and distribution of tokens will be finalized in future version of this whitepaper.

2.4 Domain name browser

NNS domain name browser is the entrance which provides NNS domain name query, auction, transfer and other functions.

2.5 Reverse resolution

NNS will support reverse resolution which will become an effective way to verify addresses and smart contracts.

2.6 Roadmap

NNS will support reverse resolution which will become an effective way to verify addresses and smart contracts.

First quarter, 2018

- January, 2018, officially released NNS technical white paper
- January, 2018, completed the technical principle test and verification
- January，31st, 2018, release the NNS Phase 1 testing service, including registrar and resolver, on the test net, anyone can register unregistered and rules-compliant domain names.
- February, 2018, launch testnet-based Domain Name Browser V1

Second quarter, 2018

- March, 2018, issue NNC on testnet.
- March, 2018, release NNS Stage 2 testing service including bidding service on testnet, when anyone can apply to NEL for NDS bidding test domain name
- April, 2018, launch testnet-based domain name browser V2.
- May, 2018, issue NNC on mainnet.
- June, 2018, release NNS service on mainnet. Here comes Neo domain name era.
- June, 2018, release mainnet-based domain name browser.

03

Design details

3.1 NNS protocol specifications

▼Protocol

The url we usually use on the Internet is as follows,

```
http://aaa.bbb.test/xxx
```

1. http is a protocol, the domain name and protocol will be passed on separately when NNS service is requested.
2. aaa.bbb.test is the domain name, NNS service request using the hash of the domain name
3. Xxx is the path, the path is not processed at the dns level, the same goes with nns, if there is a path, it will be processed by other ways.

▼Definitions of using strings in NNS protocol

The following are tentative definitions.

http protocol

http protocol points to a string, which means it's an Internet address.

addr protocol

addr protocol points to a string, which means it's a NEO address. Like:
AdzQq1DmnHq86yyDUkU3jKdHwLUe2MLAVv

script protocol

script protocol points to a byte[], which means a NEO ScriptHash. Like:
0xf3b1c99910babe5c23d0b4fd0104ee84ffeec2a5

One and the same domain name is processed differently by different protocols.

```
http://abc.test  may point to http://www.163.com
addr://abc.test  may point to AdzQq1DmnHq86yyDUkU3jKdHwLUe2MLAVv
script://abc.test  may point to 0xf3b1c99910babe5c23d0b4fd0104ee84ffeec2a5
```

3.2 A detailed explanation of NameHash algorithm

▼Namehash

The domain name NNS stores is 32byte hashes, rather than plain text of the original domain name. Here are reasons for this design.

1. A unified process allows any length of the domain name.
2. It preserves the privacy of the domain name to some extent.
3. The algorithm for converting a domain name to a hash is called NameHash,

▼Domain , domainarray and protocol

The url we usually use on the Internet is as follows,

```
http://aaa.bbb.test/xxx
```

1. http is a protocol, the domain name and protocol will be passed on separately when NNS service is requested.
2. aaa.bbb.test is the domain name, NNS service request using the hash of the domain name
3. Xxx is the path, the path is not processed at the dns level, the same goes with nns, if there is a path, it will be processed by other ways.

NNS uses domain name's array rather domain name, which is a more direct process.
Domain name aaa.bb.test is converted into byte array as ["test","bb","aa"]
You could invoke the resolution this way
NNS.ResolveFull("http",["test","bb","aa"]);

And let the contract to calculate namehash.

▼NameHash algorithm

NameHash algorithm is a way to calculate hash step by step after converting domain name into DomainArray. Its code is as follows:

```
// algorithm for converting domain names into hash
static byte[] nameHash(string domain)
{
    return SmartContract.Sha256(domain.AsByteArray());
}
static byte[] nameHashSub(byte[] roothash, string subdomain)
{
    var domain = SmartContract.Sha256(subdomain.AsByteArray()).Concat(roothash);
    return SmartContract.Sha256(domain);
}
static byte[] nameHashArray(string[] domainarray)
```

```
{
    byte[] hash = nameHash(domainarray[0]);
    for (var i = 1; i < domainarray.Length; i++)
    {
        hash = nameHashSub(hash, domainarray[i]);
    }
    return hash;
}
```

▼Quick resolution

Complete resolution introduces the whole DomainArray and let smart contracts to check every layer's resolution one by one. Calculating NameHash could also be done on Client, and then is passed into smart contracts. It's invoked this way:

```
// query http://aaa.bbb.test
var hash = nameHashArray(["test","bbb"]);// can be calculated by Client
NNS.Resolve("http",hash,"aaa");// invoke smart contracts
```

or

```
//query http://bbb.test
var hash = nameHashArray(["test","bbb"]);// can be calculated by Client
NNS.Resolve("http",hash,"");// invoke smart contracts
```

You may be thinking why querying aaa.bbb.test is not like this.

```
// query http://aaa.bbb.test
var hash = nameHashArray(["test","bbb","aaa"]);// can be calculated by Client
NNS.Resolve("http",hash,"");// invoke smart contract
```

We have to consider whether aaa.bb.test has a separate resolver. If aaa.bb.test is sold to someone else, It specifies an independent resolver so that it can be queried. If aaa.bb.test does not have a separate resolver, it is resolved by bb.test's resolver. So this cannot be queried.

The first query, regardless of whether aaa.bb.test has an independent resolver, can be found.

3.3 A detailed explanation of top-level domain name

▼Function signature of top-level domain name contracts

Function signature is as follows:

```
public static object Main(string method, object[] args)
```

Deploying adopts configuration of parameter 0710, return value 05

▼Interface of top-level domain name contract

Top-level domain name's interface is composed of three parts

Universal interface. It does not require permission verification and can be invoked by everyone.
Owner interface. It is valid only when it's invoked by the owner signature or the owner script.
Registrar interface. It's valid only when it's invoked by the registrar script.

▼Universal interface

Universal interface doesn't need permission verification. Its code is as follows.

```
if (method == "rootName")
    return rootName();
if (method == "rootNameHash")
    return rootNameHash();
if (method == "getInfo")
    return getInfo((byte[])args[0]);
if (method == "nameHash")
    return nameHash((string)args[0]);
if (method == "nameHashSub")
    return nameHashSub((byte[])args[0], (string)args[1]);
if (method == "nameHashArray")
    return nameHashArray((string[])args[0]);
if (method == "resolve")
    return resolve((string)args[0], (byte[])args[1], (string)args[2]);
if (method == "resolveFull")
    return resolveFull((string)args[0], (string[])args[1]);
```

rootName

Return the root domain name that the current top-level domain name corresponds to, its return value is string.

rootNameHash

Return NameHash the current top-level domain name corresponds to, its return values is byte[]

getInfo(byte[] namehash)

Return a domain name's information, its return value is an array as follows

```
[
  byte[] owner//owner
  byte[] register//registrar
  byte[] resolver//resolver
  BigInteger ttl//TTL
]
```

nameHash(string domain)

Convert a section of the domain name into NameHash. For example:

```
nameHash("test")
nameHash("abc")
```

Its return value is byte[]

nameHashSub(byte[] domainhash,string subdomain)

Calculate subdomain name's NameHash. For example:

```
var hash = nameHash("test");
var hashsub = nameHashSub(hash,"abc")// calculate abc.test's namehash
```

its return value is byte[]

nameHashArray(string[] nameArray)

Calculate NameArray's NameHash，aa.bb.cc.test corresponding nameArray is ["test","cc","bb","aa"]

```
var hash = nameHashArray(["test","cc","bb","aa"]);
```

resolve(string protocol,byte[] hash,string or int(0) subdomain)

resolve a domain name

The first parameter is a protocol

For example, http maps a domain name to an Internet address.

For example, addr maps a domain name to a NEO address(which is probably the most common mapping)

The second parameter is the hash of the domain name that is to be resolved.

The third parameter is the subdomain name that is to be resolved.

The following code is applied.

```
var hash = nameHashArray(["test","cc","bb","aa"]);//calculate by Client
resolve("http",hash,0)//contract resolve http://aa.bb.cc.test

or

var hash = nameHashArray(["test","cc","bb");// calculate by Client
resolve("http",hash,“aa")//smart resolve http://aa.bb.cc.test
```

The return type is byte[], how to interpret byte[] is defined by different protocols. byte[] saves strings. We will write another document to explore protocols.

Second-level domain name has to be resolved in the way of resolve("http",hash,0). Other domain names are recommended to be resolved in the way of resolve ("http",hash,“aa").

resolveFull(string protocol,string[] nameArray)

Complete model of domain name resolution

```
The first parameter is protocol
The second parameter is NameArray
```

The only difference in this resolution is it verifies step by step whether the ownership is consistent with the registration.

Its return type is the same with resolve.

▼The owner interface

All of the owner interfaces are in the form of owner_SetXXX(byte[] srcowner,-byte[] nnshash,byte[] xxx). Xxx are all scripthash.

Return value is one byte array：[0] means succeed; [1] means fail

The owner interface accepts both direct signature of account address calls and smart contract owner calls.

If the owner is a smart contract, the owner should determine their own authority. If it does not meet the conditions, please do not initiate appcall on the top-level domain contract.

▼owner_SetOwner(byte[] srcowner,byte[] nnshash,byte[] newowner)

Ownership transfer of domain names. The owner of a domain name could be either an account address or a smart contract.

srcowner is only used to verify signature when the owner is an account address. It is the address's scripthash.
nnshash is the namehash of the domain name that is to be operated.
newowner is the scripthash of new owners' address.

▼owner_SetRegister(byte[] srcowner,byte[] nnshash,byte[] newregister)

Set up Domain Registrar Contract (Domain Registrar is a smart contract) Domain Registrar parameter form must also be 0710, return 05

the following interface must be achieved.

```
public static object Main(string method, object[] args)
{
    if (method == "getSubOwner")
        return getSubOwner((byte[])args[0], (string)args[1]);
    ...

getSubOwner(byte[] nnshash,string subdomain)
```

Anyone can call the registrar's interface to check the owner of the subdomain.

There is no regulation for other interface forms of the domain name registrar. The official registrar will be explained in future documentation.

The domain name registrar achieved by the user only need to achieve getSubOwner interface.

▼owner_SetResolve(byte[] srcowner,byte[] nnshash,byte[] newresolver)

Set up a domain name resolver contract (the domain name resolver is a smart contract)

The domain name resolver's parameter form also has to be 0710 and return 05 the following interface has to be achieved.

```
public static byte[] Main(string method, object[] args)
{
    if (method == "resolve")
        return resolve((string)args[0], (byte[])args[1]);
    ...

resolve(string protocol,byte[] nnshash)
```

Anyone can call the resolver interface for resolution.

There is no regulations for other interface forms of domain name resolves. The official resolver will be explained in future documentation.

The domain name registrar achieved by the user only need to achieve resolve interface.

▼the registrar's interface

There is only one registrar interface that's called by registrar smart contract.

▼register_SetSubdomainOwner(byte[] nnshash,string subdomain,byte[] newowner,BigInteger ttl)

register a subdomain name

nnshash is the namehash of the domain names that is to be operated.

subdomain is the subdomain name that is to be operated.

newowner is the scripthash of the new owner's address.

ttl is the time to live of the domain name(block height)

If succeed, return [1], if fail, return [0]

3.4 A detailed explanation of owner contract

▼the workings of the owner contract

The owner contract calls the owner_SetXXX interface of top-level domain name contract in the form of Appcall.

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

The top-level domain name contract will check the call stack, comparing contract it's called by and the owner that manages the top-level domain name contract.

So only the owner contract of a domain name can manage this domain name.

▼the significance of the owner contract

Users could achieve complex contract ownership through the owner contract.

For example:

Owned by two persons, dual signature

Owned by more than two persons, operate by voting

3.5 A detailed explanation of registrar

▼workings of registrar contract

The registrar contract calls register_SetSubdomainOwner interface of the top-level domain name in the form of Appcall.

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

Top-level domain name contracts will check the call stack, comparing the contract it's called by and the registrar the top-level domain name contract manages.

So only the specified registrar contract can manage it.

▼the registrar interface

The registrar's parameter form also has to be 0710 and return 05

```
public static object Main(string method, object[] args)
{
    if (method == "getSubOwner")
        return getSubOwner((byte[])args[0], (string)args[1]);
    if (method == "requestSubDomain")
        return requestSubDomain((byte[])args[0], (byte[])args[1], (string)args[2]);
    ...
```

▼getSubOwner(byte[] nnshash,string subdomain)

This interface is the norms and requirements of registrars. It has to be achieved, because this interface will be invoked to verify rights when a complete resolution is conducted on the domain name.
nnshash is the hash of the domain name
subdomain is the subdomain name
Return byte[] owner's address, or blank

▼requestSubDomain(byte[] who,byte[] nnshash,string subdomain)

This interface will be used by first come first served registrar. Users call the interface of the registrar to register the domain name.
Who means who applies
nnshash means which domain name is applied
subdomain means subdomain name applied

3.6 A detailed explanation of the resolver

▼The workings of the resolver contract

1. The resolver saves resolution information by itself.
2. The top-level domain name contract calls the resolution interface of the resolver to get resolution information in the way of nep4.
3. When the resolver sets resolution data, it calls the getInfo interface of the top-level domain name contract to verify the ownership of the domain name in the way of Appcall.

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

Any contract could call the getInfo interface of the top-level domain name contract to verify the ownership of the domain name in the way of Appcall.

▼the resolver interface

The resolver's parameter form has be 0710, it returns 05.

```
public static byte[] Main(string method, object[] args)
  {
    if (method == "resolve")
      return resolve((string)args[0], (byte[])args[1]);
    if (method == "setResolveData")
      return setResolveData((byte[])args[0], (byte[])args[1], (string)args[2],
(string)args[3], (byte[])args[4]);
    ...
```

▼resolve(string protocol,byte[] nnshash)

This interface is the norms and requirements of resolvers. It's has to be achieved, because this interface will be called for final resolution when a complete resolution is conducted on a domain name.

Protocol is the string of the protocol

Nnshash nnshash is the domains name that's to be resolved.

return byte[] is to resolve the data

▼setResolveData(byte[] owner,byte[] nnshash,string or int[0] subdomain,string protocol,byte[] data)

This interface is owned by the standard resolver for demo. The owner(currently it only supports the owner of an account address) could call this interface to configure the resolution data.

owner means the owner of a domain name.

nnshash means set up which domain name

subdomain

the set-up subdomain name (could pass 0; if the set-up is domain name resolution, non-subdomain name passes 0)

protocol means the string of protocols
data means resolves data
Return [1] means succeed, or [0] means fail.

3.7 A detailed explanation of domain name registration via bid-auction

▼Bidding service

Bidding service's purpose is to determine who has the right to register a second-level domain name. this service is composed of 4 steps: opening a bid, placing a bid, revealing a bid and winning a bid.

▼Opening a bid

Any domain name that has not been registered or has expired and does not violate the domain definition can be applied by any standard address (account) to open a bid. Once the bid is opened, it means that bidding for the ownership of the domain name begins.

▼Placing a bid

Opening a bid is initiates placing a bid, which lasts for 72 hours, during which time any standard address (account) can submit an encrypted quote and pay a NNC deposit. The bidder hides the real quote by sending a sha256 hash of the binary data of a quote and a custom set of 8-bit arbitrary characters as quotes to prevent unnecessary vicious competition. If the number of bidders is less than 1 person, placing a bid automatically ends, the domain name can be immediately opened for bidding.

▼Revealing a bid

48 hours of revealing the bid comes after placing a bid is finished. During this period, bidders need to submit the quoted plaintext and encrypted string plaintext to verify the bidder's real bid. After the bid is revealed, deposit will be returned after system cost is deducted from it. The bidder who does not reveal the bid will be considered as having given up bidding. If the number of bidders is less than 1 person, the bidding ends automatically, the domain name can be opened immediately for bidding.

▼Winning the bid

After revealing the bid is finished, bid winners need to get the ownership of the domain name via a transaction.

The distribution rules of domain names via bid-auction will be specified in the future.

▼Trading service

Trading service allows domain name registrar to publish the invitation of domain name ownership transfer. It supports both fixed-price transfer and Dutch auction transfer.

3.8 Technical Realization of Lock-free cyclical redistributed token NNC

The NNS's economic system needs an asset, so we designed an asset.
The NNS's economic system requires that the total assets remain unchanged, and the auction proceeds and rental costs are considered as destroyed, so the assets we design can be consumed and the consumed assets will be redistributed, since destruction and redistribution will be cyclical, so we call it cyclically redistributed token.
Lock-free refers to the redistribution process will not lock the users' assets. The details of this will be explained below.

▼the initial distribution of tokens

NNC will be initially distributed through an ICO mechanism

▼the redistribution mechanism

We use the destruction interface to destroy tokens. Tokens to be destroyed are:

1. Rent cost will be destroyed by the system
2. Revenue from second-level domain name auction will be destroyed by the system.
3. If anyone wants to destroy part of his or her tokens, they will be destroyed by the system.

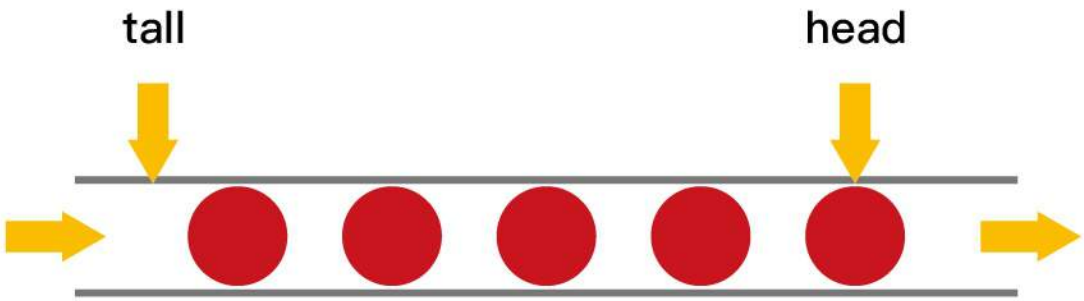
Once token are destroyed, they are counted into destruction pool. Assets in the destruction pool will go into the bonus pool, from which users could collect assets.

▼Lock-free bonus collection

Like an auction, this kind of system is usually composed of four stages: opening a bid,

placing a bid, revealing a bid and winning a bid. Users' tokens have to be paid into the system during the bidding, which means users' assets are locked, consumed after winning the bid or unlocked if the bid is missed.

However, NNC token is not composed of stages including participating bonus collection, waiting for the bonus and collecting the bonus, which means users' assets are not locked in the whole process.



NNC uses the mechanism of the bonus pool queue, as shown in the above picture, only a fixed number of bonus pools (for example, five) are kept.
The oldest bonus pool(the head pool)will be destroyed when more than five bonus pools are generated.

Besides the bonus pool, users' assets are composed of two types: fixed assets and change. The holding time of fixed assets can only increase, and users whose holding time is earlier than collection time of a bonus pool are qualified to collect bonus.

The holding time of fixed assets increases after collecting the bonus. It's like coin hours is consumed, thus preventing repeated collection of the same bonus.

▼Details of the bonus pool

The token will maintain several bonus pools. When each bonus pool is generated, the assets in the destruction pool will all be transferred into this bonus pool. If the maximum bonus pool number is exceeded, the oldest bonus pool will also be destroyed and the remaining assets in the destroyed bonus pool will also be counted in the latest bonus pool.

The number of bonus pools is fixed, for example, a bonus pool is generated for every 4096 blocks. A maximum of five bonus pools are maintained. When the sixth bonus pool is generated, the first bonus pool will be destroyed, and all of its assets are placed in the latest bonus pool. The above number of bonus pools and how often one bonus pool is produced are both tentative)

Each bonus pool will correspond to a block, this block is the bonus collection time, only those whose holding time is earlier than the bonus collection time can collect the bonus.

▼Details of the bonus pool

The token will maintain several bonus pools. When each bonus pool is generated, the assets in the destruction pool will all be transferred into this bonus pool. If the maximum bonus pool number is exceeded, the oldest bonus pool will also be destroyed and the remaining assets in the destroyed bonus pool will also be counted in the latest bonus pool.

The number of bonus pools is fixed, for example, a bonus pool is generated for every 4096 blocks. A maximum of five bonus pools are maintained. When the sixth bonus pool is generated, the first bonus pool will be destroyed, and all of its assets are placed in the latest bonus pool. The above number of bonus pools and how often one bonus pool is produced are both tentative)

Each bonus pool will correspond to a block, this block is the bonus collection time, only those whose holding time is earlier than the bonus collection time can collect the bonus.

▼Details of fixed assets and chang

Fixed assets and change, of which fixed assets record a holding time.
Fixed assets and change only affect the amount of the award, the rest of the functions are not affected.
Fixed assets + change = user's balance of assets
Fixed assets do not have a fractional part, the decimal part is counted in the change.
When " considered as fixed assets" is mentioned below, it means the integer part is considered as fixed assets, and decimal part as change.

Change will be firstly used in transfer of tokens and fixed assets will be used only when the change is not enough.

Transferrer: fixed assets can only be reduced.

Transferee: fixed assets remain unchanged, transferred value is counted in the change.

Fixed assets can only be increased in two ways.

1.Create an account (a transfer to an address which has no NNC is regarded as creating an account)

The transferred assets are regarded as fixed assets and the holding time is the new block ID.

2.Collect the bonus

After collecting the bonus, personal assets and the collected bonus will be considered as fixed assets, holding time is the bonus block.

When collecting bonus, users can only collect bonus when their holding time is earlier than bonus pool time.

Bonus collection ratio is calculated as the total amount in the current bonus pools/(the total issuance volume-the total amount in the current bonus pools)

Let's take numbers to exemplify it. For example, there are 3 bonus pools: they were produced by block 4096，8000，10000. One user's fixed assets is 100. His holding time is 7000, then he cannot collect the bonus in the first pool, but can collect bonus from the second and third pools. The current block is 10500. Once the user collects the bonus, his assets holding time becomes 10500, so he cannot collect bonus from any pools.

For example, there is 50300000 NNC in a bonus pool. Then the user's bonus collection ratio is 50300000 /（100000000-50300000）=1.23587223. This user's fixed asset is 100, then he can collect 123.587223 NNC from the pool.

If there is 500, 000 NNC in a bonus pool, then his collection ratio is 500000 /（100000000-500000）=0.00502512, as the user has 100 NNC of fixed asset, then he can collect 0.502512 NNC from the pool.

▼NNC interface(only additional interfaces compared with NEP 5 will be described)

NNC first meets the NEP5 standard, and the NEP standard interface will not be described any more.

·balanceOfDetail(byte[] address)

Returning the details of the user's assets, such as how much fixed assets, how much change, the total amount. Fixed assets holding block does not need a signature. Anybody can check it.

}

Return structure

{

Cash amount
The amount of fixed assets
Fixed assets generation time(new block ID)
Balance (fixed assets + cash)
}

·use(byte[] address,BigInteger value)

The consumption of assets in an account requires the account signature.
Consumed assets go into bonus pools.

·getBonus(byte[] address)

Account signature is required when designated accounts collect the bonus.
After the collection of the bonus, the total assets in this account will be considered as fixed assets and fixed assets holding block of this account will be changed.

·checkBonus()

Checking current bonus pool doesn't need a signature.
Return Array<BonusInfo>

BonusInfo
{
StartBlock;//bonus collecting block
BonusCount;//total amount of this bonus pool
BonusValue;// remaining amount of this bonus pool.
LastIndex;// the id of last bonus
}

·newBonus ()

Generating a new bonus pool can be called by anyone. But the bonus pool generation has to meet the bonus pool interval, so repeated calls is useless. This interface can be seen as a push to generate a new bonus pool.

04

Summary

The NNS project is a smart contract protocol layer application built entirely on the NEO blockchain and is a true blockchain application. The combination of many projects and blockchain is just to issue a kind of token, the service of the token is provided by a central organization, while all services of the NNS are provided by the smart contract, which is distributed and flexible and extensible, without centralization risks. NNS is a large-scale application of NEO smart contract system. In order to achieve resolver flexibility and scalability, we apply the latest NEP4 dynamic call. In the economic model, we will design the Vickrey auction contract and the Dutch auction contract. In order to achieve an easy redistribution of system costs, we've extended the NEP5 tokens standard and added the concept of coin days to allow system revenue redistribution without locking tokens, blending application tokens and equity into one kind of token.

As domain name services can improve the usability of the blockchain and have rich usage scenarios, there will form an ecosystem around domain names. In the future, we will work with NEO Eco-system C-lients to allow all NEO wallets to support the transfer of tokens via alias. We will also explore new usage scenarios such as working with pet games and giving pets nicknames via NNS. In the future, as the NEO ecosystem is used more and more, the domain name of NNS will become more and more valuable.

05

3423

References

[1]. NEO White Paper: Superconducting Exchange. https://github.com/neo-project/docs/blob/e01d268426a8b5f9b3676cfd03d0b8b83d7711a1/en-us/white-paper.md#highly-scalable-architecture-design, Accessed 2018.

[2]. NEO NEP-5: Token Standard. https://github.com/neo-project/proposals/blob/master/nep-5.mediawiki, Accessed 2018.

[3]. ENS. https://github.com/ethereum/ens,Accessed 2018.

[4]. EIP137 - Ethereum Name Service. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-137.md，Accessed 2018.

[5]. EIP162 - Initial ENS Registrar Specification. https://github.com/ethereum/EIPs/issues/162，Accessed 2018.

Contact information

Mail：contact@neons.name

Website：neons.name

Twitter：@NewEconoLab

QQ group：702673768

WeChat public number：NEL新经济实验室