Nebulas Technical White Paper

The value-based blockchain operating system and search engine

Nebulas Team

September, 2017
v1.0

Abstract

Bitcoin and Ethereum have successfully introduced "Peer-to-Peer Electronic Cash System" and "Smart Contract" to blockchains. The industry is evolving rapidly, with emerging application scenarios and business requirements. For current blockchain technologies, we find there are three challenges: measure of value, self-evolving capability, and healthy ecosystem development.

Nebulas aims to address those challenges. This white paper explains the technical design ideologies and principles of the Nebulas framework. The framework includes:

- **Nebulas Rank (NR)** (§2), which measures value by considering liquidity and propagation of the address. Nebulas Ranking tries to establish a trustful, computable and deterministic measurement approach. With the value ranking system, we will see more and more outstanding applications surfacing on the Nebulas platform.
- **Nebulas Force (NF)** (§3), which supports upgrading core protocols and smart contracts on the chains. It provides self-evolving capabilities to Nebulas system and its applications. With Nebulas Force, developers can build rich applications in fast iterations, and the applications can dynamically adapt to community or market changes.
- **Developer Incentive Protocol (DIP)** (§4), designed to build the blockchain ecosystem in a better way. The Nebulas token incentives will help top developers to create more values in Nebulas.
- **Proof of Devotion (PoD) Consensus Algorithm** (§5). To build a healthy ecosystem, Nebulas proposes three key points for consensus algorithm: speediness, irreversibility and fairness. By adopting the advantages of PoS and PoI, and leveraging NR, PoD will take the lead in consensus algorithms.
- **Search engine for decentralized applications** (§6). Nebulas constructs a search engine for decentralized applications based on Nebulas value ranking. Using this engine, users can easily find desired decentralized applications from the massive market.

Contents

1 Introduction **6**

1.1 Blockchain technology introduction . . . 6

1.2 Business and technology challenges . . . 6

1.3 Nebulas design principles . . . 7

2 Nebulas Rank **9**

2.1 Overview of Nebulas Rank . . . 9

2.2 Transaction Graph . . . 10

2.3 Ranking Algorithm . . . 13

2.4 Manipulation Resistance . . . 14

2.5 Related Works . . . 19

3 Nebulas Force **21**

3.1 Nebulas Virtual Machine (NVM) . . . 21

3.2 Upgrade Design of the Protocol Code . . . 23

3.2.1 Block Structure . . . 23

3.2.2 Upgrade of the Protocol Code . . . 25

3.3 Upgrade Design of the Smart Contract . . . 26

3.3.1 Turing Complete Smart Contract Programming Language . . . 26

3.3.2 Upgradable Contracts . . . 27

4 Developer Incentive Protocol (DIP) **30**

4.1 Design Goals . . . 30

4.2 DIP Reward Distribution Algorithm . . . 30

4.3 Experimental Results . . . 31

4.4 Cheating Analysis . . . 32

5 Proof of Devotion (PoD) Consensus Algorithm **33**

5.1 Design Goals . . . 33

5.2 Defects of Commonly Used Consensus Algorithms . . . 33

5.3 Design of the PoD Algorithm . . . 34

5.3.1 Generation of New Blocks . . . 34

5.3.2 Dynamic Validators Set . . . 34

5.3.3 Consensus Process . . . 35

5.3.4 Fork Choice . . . 36

5.3.5 Slashing Conditions . . . 36

5.4 PoD Economic Analysis . . . 37

5.4.1 Incentive Analysis . . . 37
5.4.2 Cheating Analysis . . . 37

6 Blockchain Search Engine **41**
6.1 Introduction . . . 41
6.2 Infrastructure . . . 41
6.3 Nebulas Trends . . . 43
6.4 Keyword Query . . . 44
6.5 Searching for Similar Smart Contracts . . . 44

7 Fundamental Services and Development Tools **45**
7.1 Domain Name Service . . . 45
7.2 Lightning Network . . . 45
7.3 Developer Tools . . . 46

8 Nebulas Token NAS **47**

9 Conclusion **48**

Appendix A Nebulas Account Address Design **55**
A.1 Address Checksum . . . 55
A.2 Extended Address Verification . . . 55

Appendix B Search for Similar Smart Contracts **57**

Glossaries

BFT	Byzantine Fault Tolerant
DIP	Developer Incentive Protocol
NF	Nebulas Force
NNS	Nebulas Name Service
NR	Nebulas Rank
NVM	Nebulas Virtual Machine
PoD	Proof of Devotion
PoI	Proof of Importance
PoS	Proof of Stake
PoW	Proof of Work
SCR	Smart Contract Rank
SCS	Smart Contract Score
WAA	Weekly Active Addresses

1 Introduction

1.1 Blockchain technology introduction

Blockchain technology is derived from a decentralized digital currency, Bitcoin, which was conceptualized by Satoshi Nakamoto in 2008 [37]. Instead of being issued by any institutions, Bitcoin is generated through specific algorithm and massive computing to ensure the consistency of the distributed ledger system. Ethereum [10] goes further and provides a public blockchain-based computing platform with a Turing-complete language. The core of cryptocurrency systems represented by Bitcoin and Ethereum is blockchain technology. With data encryption, timestamp, distributed consensus and economic incentive, blockchain technology brings into reality peer-to-peer transaction, coordination and collaboration in the distributed system in which the nodes do not need to trust each other, resolving common issues faced by centralized institutions including high cost, low efficiency and insecure data storage.

It should be noted that the blockchain technology itself is not a brand new technological innovation, but a model innovation that combines a series of technologies including peer-to-peer communication, cryptography, block-chain data structure, etc.

1.2 Business and technology challenges

As more people join in the development camp of decentralized self-governing systems represented by blockchain technology, the world has seen a dramatic increase of blockchain projects to over 2,000 with the value of global encrypted digital assets amounting to $90 billion dollars. The number of blockchain users/digital asset owners is also rising rapidly from 2 million in the beginning of 2013 to 20 million in early 2017. By 2020, the number of blockchain users/digital asset owners is expected to reach or surpass 200 million, and by 2025, 1 billion.

With the popularity of blockchain technology, more blockchain-based application scenarios are emerging. Such scenarios have gradually extended from digital currency to a broader range and user-group coverage. For example, Ethereum community introduced the concept of smart contract to blockchain technology; Ripple implemented a global settlement system using blockchain technology. Diverse application scenarios also come with increasing demands and challenges.

Lack of measure of value. We believe that the blockchain world needs a universal measure of value to measure the value of users and smart contracts. Upper-layer applications can be built on this universal measure of value to seek deeper value in its particular scenario. In this sense, innovations in business models will abound in the future, reminiscent of Google's rise in the world of Internet.

Blockchain system upgrade. Unlike common software, the decentralized blockchain system cannot enforce users to upgrade clients and protocols. Therefore, protocol upgrade in the blockchain system

often leads to "hard fork" or "soft fork" and results in huge losses, which further limits the application of the system. In the case of Bitcoin, controversy still abounds over block scaling within the community, which hinders the evolution of Bitcoin protocols. The severe undercapacity of block has once led to a situation where nearly one million transactions were waiting in the transaction pool to be written into the blocks. Users often have to pay an extra high "transaction acceleration fee", which seriously affect user experience. Moreover, although Ethereum's "hard fork" offers a temporary fix to The DAO problem, it also gives rise to "side effects" like ETH and ETC "duplicate assets" and division of community.

The construction of blockchain application ecosystem. With applications (DApp) increasing rapidly on the blockchain, a sound ecosystem becomes the key to better user experience. What we should think about is how to help users search and find the desired DApp in a massive collection of blockchain applications, how to encourage developers to develop more DApps for users, and how to assist developers with the construction of DApps. Take Ethereum for example. Hundreds of thousands of apps have been built on Ethereum now, and when the number of blockchain DApps reaches that of Apple App Store, how to search and find the expected DApp would be a big challenge.

1.3 Nebulas design principles

We set out to design an incentive-based and self-evolving blockchain system to take up those challenges and opportunities. The design principles are as follows:

- **A fair ranking algorithm to define the measure of value**

  We believe that the blockchain world needs a universal measure of value to measure the value of simple data at the bottom layer of blockchain to identify a higher dimension of information, thus exploring greater value of the blockchain world. We put forward the NR (Nebulas Rank) (see §2) algorithm (similar to Google's PageRank [9][45]), which combines the liquidity, speed, width and depth of capital on the blockchain to provide a fair ranking for blockchain users. NR is the measure of value in the blockchain world, with which developers can measure the importance of each user, smart contract, and DApp in different scenarios. NR has huge commercial potential and can be used in search, recommendation, advertising and other fields.

- **The self-evolving of blockchain system and applications**

  We believe that a well-conditioned system and the applications on it should be able to self-evolving, which means to achieve faster computing, better resilience, and enhanced user experience under little intervention. We call this self-evolving ability "Nebulas Force" (see §3). In Nebulas'system architecture, thanks to our well-designed block structure, base protocols will become part of the data on the blockchain and achieve upgrade through the addition of data. As for applications (smart contracts) on Nebulas, we make the upgrade of smart contracts possible by enabling cross-contract access to state variables at the bottom layer storage of smart contracts. The self-evolving Nebulas will be advantageous over other public blockchains in terms of developmental and survival

potential. It also allows developers to respond faster to loopholes with upgrades and prevents huge losses caused by hacking.

- **The construction of blockchain application ecosystem**

  We develop the PoD (see §5) algorithm based on the devotion of accounts on Nebulas. This algorithm uses NR as the measure of value to identify the accounts with great devotion to the ecosystem, and grant them the equal probability to be bookkeeper on an equal basis to curb the monopoly in bookkeeping. It also integrates the economic penalties in PoS to prevent malicious damage to public blockchains, facilitating the freedom of ecosystem. Featured by faster consensus speed and stronger anti-cheat ability than PoS and PoI, PoD is a positive force for the development of blockchain ecosystem.

  We also develop the DIP (Developer Incentive Protocol) (see §4) for smart contracts and DApp developers, which aims to reward smart contract or DApp developers for their great devotion to the community. The incentive is written into the blockchain by the bookkeeper. Based on the Nebulas Rank mechanism, Nebulas further includes a search engine (see §6) to help users better explore high-value applications in the blockchain.

Since Ethereum is a successful public blockchain platform with an ecosystem of massive scale, Nebulas hopes to learn from Ethereum's excellent design, and make smart contract fully compatible with it, so that Ethereum-based applications can run on Nebulas with zero migration cost.

Based on the above principles of design, we strive to build a blockchain operating system and a search engine based on the measure of value. This white paper describes in detail the technologies embedded in Nebulas. §2 explains the Nebulas Rank, a model of measure of value, and its algorithm; §3 describes the Nebulas Force, a self-evolving capability of Nebulas; §4, §5, §6, §7 are about Nebulas'conception and design of the ecosystem for blockchain applications; and §8 discusses NAS, the token of Nebulas.

2 Nebulas Rank

2.1 Overview of Nebulas Rank

Currently the Blockchain technology and community have grown into a large scale ecosystem. However, people's perception of Blockchain world is still relatively flat; there is no reasonable way to evaluate the value of an entity (such as user address) on the blockchain yet. Therefore, we try to come up with a universal measure of value. By exploring and utilizing activities on chain, we create **Nebulas Rank** through which the value of each entity (user address) is able to be quantified. **Nebulas Rank** is designed to:

- serve as a native measure of value and a core algorithm for many fundamental scenarios, such as consensus algorithm (see §5), DIP (see §4) and Blockchain search engine (see §6), etc;
- inspire diversified measures of value and deeper insights into the blockchain ecosystem, so as to better guide business decisions and research activities.

Based on the goals above, we define the measure of value of **Nebulas Rank** from three dimensions:

- **Liquidity**, the frequency and scale of transactions, is the first dimension that **Nebulas Rank** measures. Finance essentially is about the social activities which optimize social resources via capital liquidity and promote economic development. Blockchain establishes a network of value. More transactions and larger transaction scale produce better liquidity, and better liquidity further increases transactions and transaction scale, forming a complete mechanism of positive feedback.
- **Propagation**, the scope and depth of asset liquidity, is the second dimension that **Nebulas Rank** measures. In social network, the propagation property, i.e. speed, scope and depth of information transmission, is a key index indicating network quality and user growth. We see the same pattern in the Blockchain world. Powerful propagation means wider and deeper asset liquidity, which improves the quality and scale of assets in the Blockchain world.
- **Interoperability** is the third dimension that **Nebulas Rank** measures. During the early stage of Internet, there were just simple websites and isolated information. Now information on different platforms can be forwarded on the network, and isolated data silos are gradually being broken. This tendency could be understood as a process of recognizing information from higher dimensional perspective. We believe that Blockchain world also follows a similar pattern, whose development will be faster. There will be more information of users' assets, smart contracts and DApp. And also, there will be more frequent interactions among them. Therefore, better interoperability will become more important.

We choose transaction records on chain as source data for **Nebulas Rank**. That is because the "trajectory" in Blockchain world is more clear and trustworthy than that in the real world. Transaction data such as transfers and callings of "smart contracts" are all recorded on chain. But it is not trivial

to design rank algorithm for Blockchain transaction data, since the transactions in Blockchain world are naturally anonymous and bear larger data scale than that in the real world. Three major properties for **Nebulas Rank** are described below:

- Truthful. An entity must pay reasonable costs to improve its ranking, which assures that the algorithm can identify trusted valuable users. On one hand, in scenarios like consensus algorithm and DIP, truthful ranking encourages users to contribute truthfully in order to realize positive feedback. On the other hand, truthful result provides meaningful hierarchy representation of all users, which will be more helpful for decision makers;
- Computable. As a fundamental field, the result of **Nebulas Rank** should be accessible instantly and thus requires low computational complexity;
- Deterministic. As required by scenarios such as consensus algorithm and DIP, the computing result of **Nebulas Rank** should be identical on any client.

Next we design the basic framework of **Nebulas Rank**. First, transaction records are represented in the form of graph. In the transaction graph (entity graph), every node is mapped to one entity, and each edge represents the transfer between two entities[56]. Transaction graph embodies the fact that money transfer among users leads to assets flowing, which helps to represent the concepts of liquidity and propagation defined before. Meanwhile, the form of graph can clearly display the interoperability among contracts. With the derived transaction graph, we rank nodes by their network centrality. In the scenario of **Nebulas Rank**, LeaderRank[14][31] is a more reasonable measurement and outperforms PageRank of Google and NCDAwareRank of NEM[38].

2.2 Transaction Graph

This subsection introduces how to derive transaction graph from transaction history.

First, for any time t_0, we take all **effective** transactions among individual addresses during $[t_0 - T, t_0]$ (T is a constant, generally set to one month), where every transaction can be represented as a 4-tuple (s, r, a, t), with s as the source address, r as the target address, a as the transfer amount and t as the block time. We define a transaction to be **effective** iff $a > 0$ and $s \neq r$. Thus all **effective** transactions during $[t_0 - T, t_0]$ can be represented by a set of 4-tuples:

$$\Theta(t_0) = \{(s, r, a, t) \mid t_0 - T \leq t \leq t_0 \ \wedge \ a > 0 \wedge s \neq t\} \tag{1}$$

Then based on $\Theta(t_0)$, construct a directed weighted simple graph $G = (V, E, W)$, where node set, edge set and edge weights are denoted by V, E and W respectively. Every node in V represents an individual account's address, and each edge in E represents the transfer intensity between two addresses. Edges are directed and are assigned with weight w_e aggregating top K amounts of all related transactions:

$$w_e = \sum_{i=1}^{K} a_i, \ s.t. \ a_i \in A_e \tag{2}$$

A_e is an ordered set consisting of amounts of all transactions from s to r in $\Theta(t_0)$:

$$A_e = \{a_i \mid e = (s, r) \wedge (s, r, a_i, t) \in \Theta(t_0) \wedge (a_i \geq a_j, \forall i \leq j)\} \tag{3}$$

Additionally, let $N = |V|,\ M = |E|$, where $|.|$ is the cardinal number of a set. For simplicity, every node is represented by an integer between 1 and N.

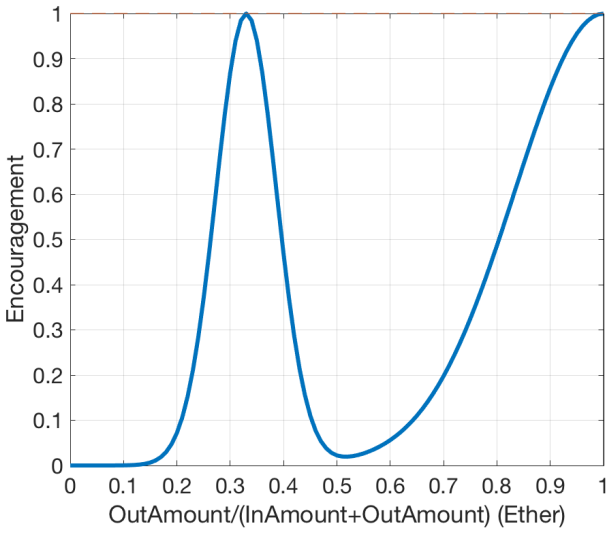


Fig. 1: Encouragement Function

Then, for each node, according to its in-transfers and out-transfers during $[t_0 - T,\ t_0]$, compute the "coinage" and denote it by C_v; based on the total transfer amount and using the "encouragement function" shown in Fig.1, compute the encouragement value and denote it by E_v[1]; use C_v and E_v of the target nodes to reduce the edges' weight.

Finally, take the largest weakly connected component of the whole graph, only keep nodes belong to this component. The deleted nodes are assigned with lowest importance score by default.

The graph derivation described above contributes to the "truthful" property defined at §2.1, the evidence of which is shown in §2.4.

Using the method described herein, the transaction graph, wherein T is 30 days and $K = 2$, is created on the basis of transaction data in the main chain of Ethereum, from block #3629091(roughly on May 1st, 2017) to block #3800775(roughly on May 31st, 2017). Its visualization is shown as Fig.2. , and all nodes are resized by their degree. It could be observed that some famous exchanges usually interacts with more accounts than others. Besides, the identities of some addresses who contribute a lot transactions still remains unknown.

[1]Encouragement function can be represented as a linear combination of two normal distributions, which outputs peak value when the money transferred out is zero or of some ratio of the amount transferred in.

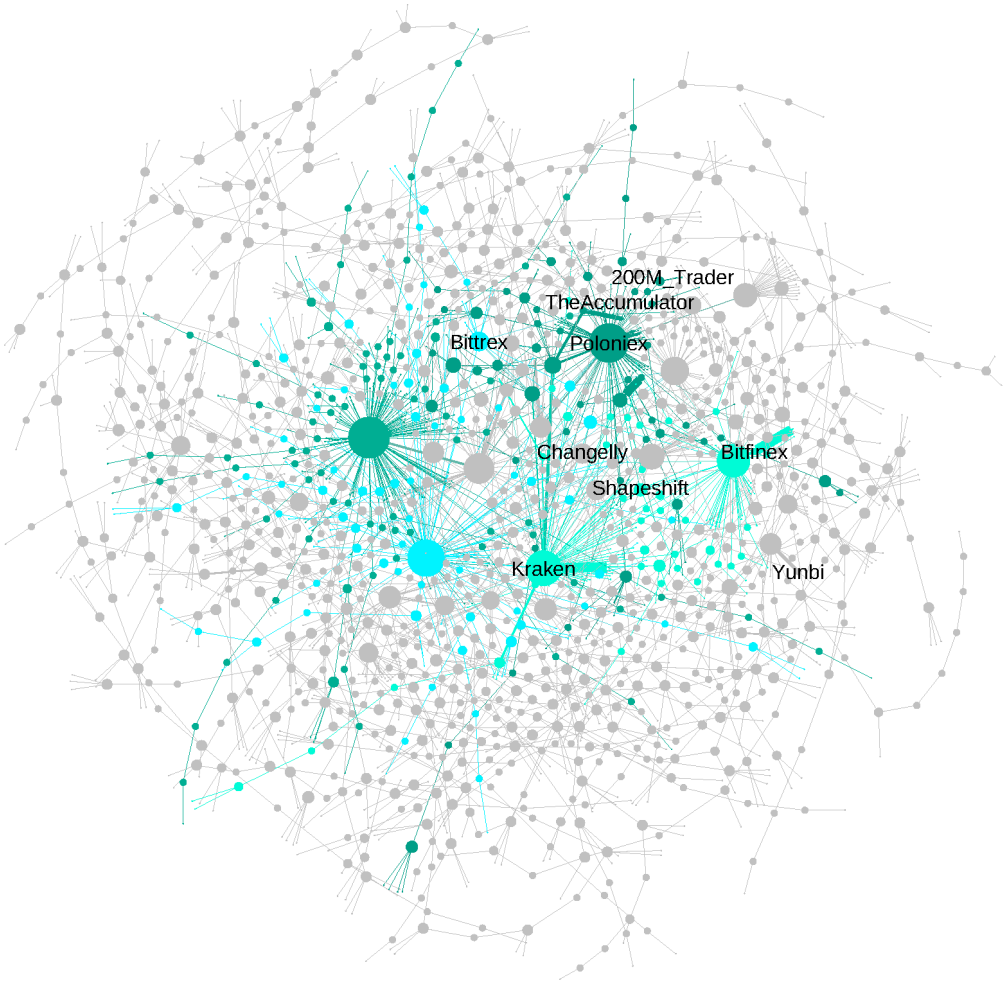


Fig. 2: Transaction Graph (Partial) Visulazation. Large transaction scale (capital transfer) in the address means high in-and-out degree in the node, represented by large diameter in the figure. Some nodes are labeled by tags according to Etherscan[16]

2.3 Ranking Algorithm

This subsection introduces how to rank nodes by their importance in the derived transaction graph.

We adopt LeaderRank[14][31] as the main algorithm. First, add a ground node with index 0 into the transaction graph. Then establish bi-directional link between the ground node 0 and every other node i $(1 \leq i \leq N)$, weighting by the following formula:

$$w_{i,0} = \alpha(max\{\sum_{w_{j,i}\neq 0} w_{j,i} - \sum_{w_{i,j}\neq 0} w_{i,j}, 0\} + \lambda C), \forall i \in [1, N] \tag{4}$$

$$w_{0,i} = \beta(\sum_{w_{i,m}\neq 0} w_{i,m} + \mu C), \forall i \in [1, N] \tag{5}$$

C is the median of set $\{w_{i,j}|w_{i,j} \neq 0, 0 \leq i, j \leq N\}$, and $\alpha, \beta, \mu, \lambda$ are parameters.

The weighting scheme can be explained that, nodes with more in-degree receive more in-link from the ground node; nodes with more absolute income, i.e. in-degree minus out-degree, outputs stronger link into the ground node.

The computing process of LeaderRank is similar to PageRank, which could be understood as computing the convergence state of a Markov process. The only difference is that, after adding the ground node, it does not need to consider the damping factor of PageRank[9][45] anymore. That is, after constructing matrix H according to formula (7), the computing process is iterated until convergence, as formula (6) shows, with initial settings defined by formula (8). Finally the rank score of ground node is distributed evenly to every other node, which yields the final score for every node.

$$P^{t+1} = H \times P^t; P^1 = [0, \frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}]^T \tag{6}$$

$$h_{ij} = \frac{w_{(j\ i)}}{\sum_k w_{(j,k)}} \tag{7}$$

$$\forall v \in V, P_v^* \leftarrow P_v^* + \frac{P_{\mathcal{G}}^*}{N} \tag{8}$$

We suppose that LeaderRank can satisfy the measure of value and algorithm property defined in §2.1.

- The result of LeaderRank can be understood as the flux on each node in the dynamic equilibrium of money exchange network, which matches **Nebulas Rank**'s measure of value: "liquidity", "propagation" and "interoperability";
- The weighing scheme defined by formula (5) and (4) makes it more difficult to attack (see discussion in §2.4), which satisfies "truthful" property;
- LeaderRank can be computed by power iteration. Since the network is very sparse, the complexity of matrix computation should not be high, which satisfies the properties of "computable" and "deterministic".

2.4 Manipulation Resistance

Truthfulness, i.e. the ability of resisting manipulation, is the most significant and challenging goal of **Nebulas Rank**. Some manipulation methods are as follows:

1. Loop transfer. The attacker transfers along a loop topology, which allows the same money flow over same edges repeatedly. By this means, the attacker hopes to raise the weight of related edges;
2. Transfer to random addresses, so that the out-degree of sybil node is increased, and the propagation of fund is increased as well;
3. Form an independent network component with addresses controlled by the attacker. So the attacker can pretend to be a central node;
4. Interact with authoritative Exchange service addresses frequently, i.e. transfer the same money in and out an authoritative Exchange service address repeatedly, so that the attacker can acquire better structural position in the network.

Nebulas Rank mitigates manipulation through the following mechanism:

- Owing to sliding windows of T blocks, the attacker cannot increase its rank in short term;
- Since the edge weight is decided by the related transactions with highest amount , transferring along a loop topology cannot increase edges' weight unlimitedly. Meanwhile, according to the sampled data in §2.2, 91% of edges correspond to less than 2 transactions respectively. Thus $K = 2$ is a reasonable choice to remain the intensity information on edges while being resistance to loop transfer;
- In order to have higher "coinage", the user needs to let money stay in their address for a while, which slows down the attacking speed;
- In order to get the maximum "encouragement value", as is shown in Fig.1, the account needs to keep all income or transfer out only a small ratio of it. So when forging money flow, the attacker will get a rapid decrease in its deposit;
- Because only the giant component is selected, other independent components including the forged one will be filtered out as noise. According to the sampled data in §2.2, there are $453,285$ nodes and $970,577$ edges, with $1,169$ components. In the biggest component, there are $449,746$ nodes, accounting for 99.2% of the total number. In the second biggest component, there are just 133 nodes, only accounting for 0.03%. Thus taking the giant component can remain normal part of network as much as possible while filtering noise part out;
- Compared with webpage ranking algorithms such as PageRank and NCDawareRank[42], the mechanism defined by formula (5) and (4) is more "conservative" on the nodes with low income, i.e. nodes with low in-degree get weaker links from the Ground node. In the blockchain transaction graph, nodes with low income are more likely to be generated, and transferring to other random nodes cannot raise in-degree. So **Nebulas Rank** can increase the difficulty of manipulation.

Next, the following conclusions are made based on the transaction graph of Ethereum in May, 2017.

First, some addresses by **Nebulas Rank** are listed in table 1[2]. It can be observed that the Exchange addresses and some accounts with high transaction throughput are ranked as top nodes.

Table 1: **Top 10** addresses of **Nebulas Rank** and some other addresses

Rank (Order)	Address	Nebulas Rank	Domain	Out Amount(Ether)	In Amount(Ether)
1	0x267be1c1d684f78cb4f6a176c4911b741e4ffdc0	0.449275	Kraken_4	3214232.06	350008.00
2	0xd4c5867cec094721aabc3c4d0fd2f2ac7878c79a	0.093798		58000.00	100947.00
3	0x027beefcbad782faf69fad12dee97ed894c68549	0.049277	QuadrigaCX	207440.11	65606.40
4	0x0ee4e2d09aec35bdf08083b649033ac0a41aa75e	0.046831		56465.00	60087.96
5	0xc257274276a4e539741ca11b590b9447b26a8051	0.037628		1071105.93	1434106.72
6	0xa53e0ca7d246a764993f010d1fde4ad01189f4e6	0.033488		7764.68	3201.00
7	0xf259e51f791e9ed26e89b6cae4a7c6296bfbd0b8	0.033481		3307.00	7731.30
8	0xf195cac8452bcbc836a4d32cfb22235af4ac1e9c	0.026343		10863.87	2315.69
9	0x94435d12c51e19d5b5c8656763f9069d37791a1a	0.024970		12938.58	15858.90
10	0x7580ba923c01783115d79975d6a41b3d38eff8d5	0.021670		263000.00	364793.49
16	0xcafb10ee663f465f9d10588ac44ed20ed608c11e	0.004995	Bitfinex_1	360000.00	1435858.40
51	0xd94c9ff168dc6aebf9b6cc86deff54f3fb0afc33	0.000868	yunbi_1	1179224.74	1202539.53
64	0x70faa28a6b8d6829a4b1e649d26ec9a2a39ba413	0.000590	Shapeshift	52501.81	651933.49

[2]Source of domain: Etherscan[16]

Then, we observe the relationship between transaction amount and **Nebulas Rank**. Since blockchain transferring can be understood as network flow of "money exchange", according to Borgatti [5]'s work, the degree of nodes, i.e. sum of adjacent edges' weights, is a proper centrality metric for such network flow. From the perspective of each node, the degree, i.e. transaction throughput amount (amount transferred in plus amount transferred out), represents local information within one hop and reflects the historical money flow over corresponding addresses. So it could be a baseline for ranking algorithms. The relationship between transaction amount and **Nebulas Rank** is shown in Fig.3: no node can acquire high rank with low transaction amount; nodes with high transaction amount still need to meet some conditions to get high rank, which roughly confirms the truthfulness of **Nebulas Rank**.

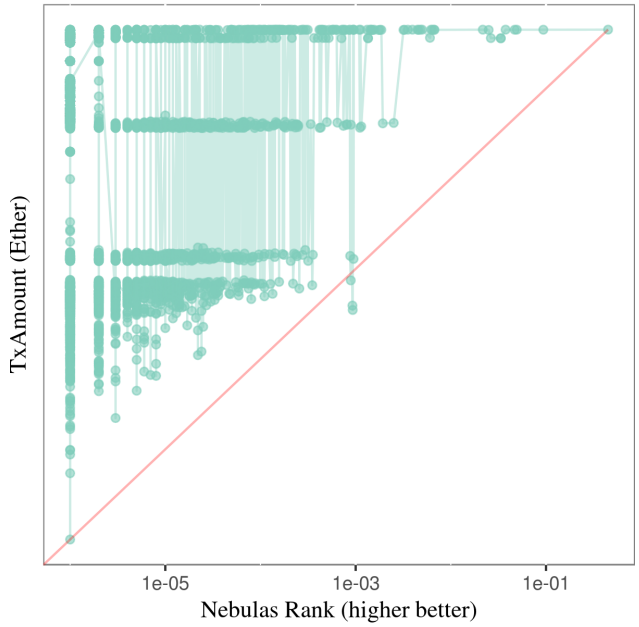


Fig. 3: Nebulas Rank v.s. Transaction Amount X-axis represents Rank Value, and Y-axis represents Transaction Amount, both in logarithmic form. The diagonal line represents that transaction amount and rank value is directly proportional. A good algorithm should make the data points fall as little as possible in the lower right of the diagonal line, to avoid nodes with low transaction throughput to get high rank.

According to the above simple analysis, it can be deduced that the first three types of manipulation can be filtered effectively by specific methods. Therefore finally, we just need to simulate the last type and observe the resistance effect. The attacker chooses one authoritative Exchange node to create loop transfers for X times. Every loop transfer contains 2 phases: first the attacker transfers Y Ether to the Exchange through some newly created temporary address; then the attacker retrieves its money back from the Exchange through another address. The attacking topology and process is demonstrated in Fig.4. This type of attack exploits the fact that an Exchange service is willing to establish links with any nodes at a very low cost. Although normal nodes can also transfer with Exchange nodes frequently,

but such attacking activities do not improve effective money liquidity and should be distinguished from normal activities.

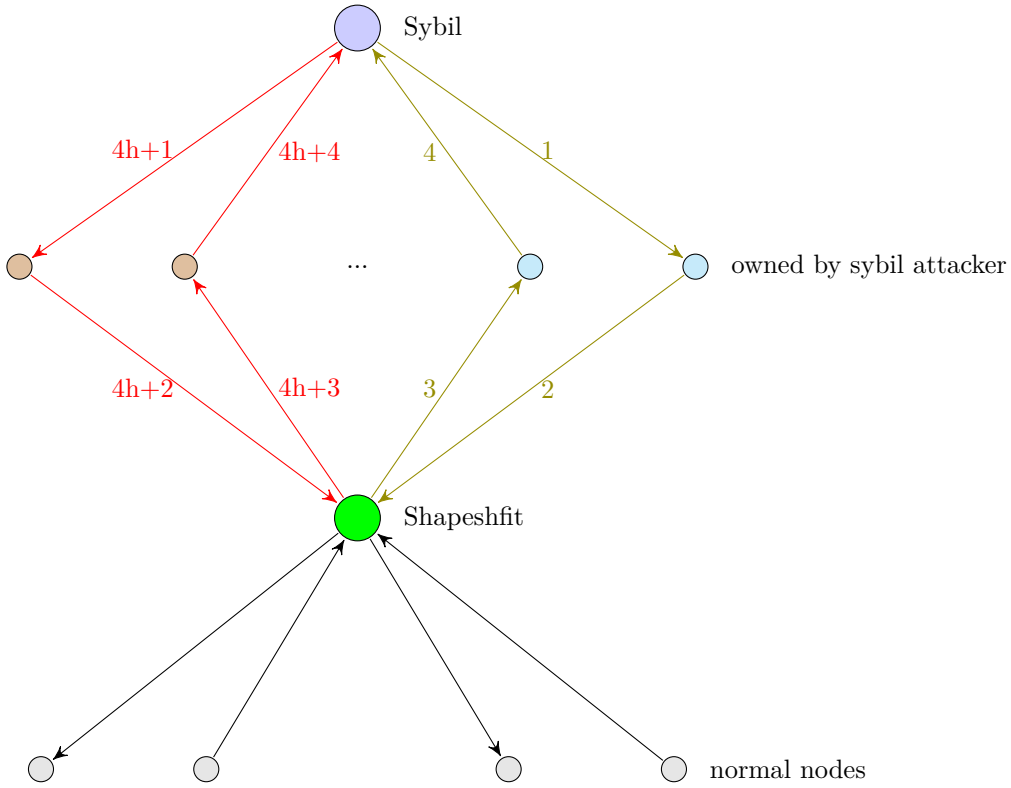
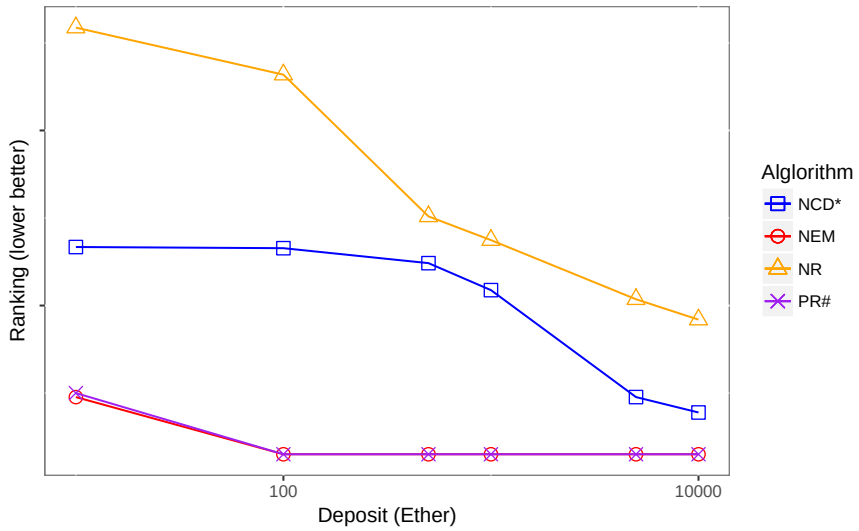
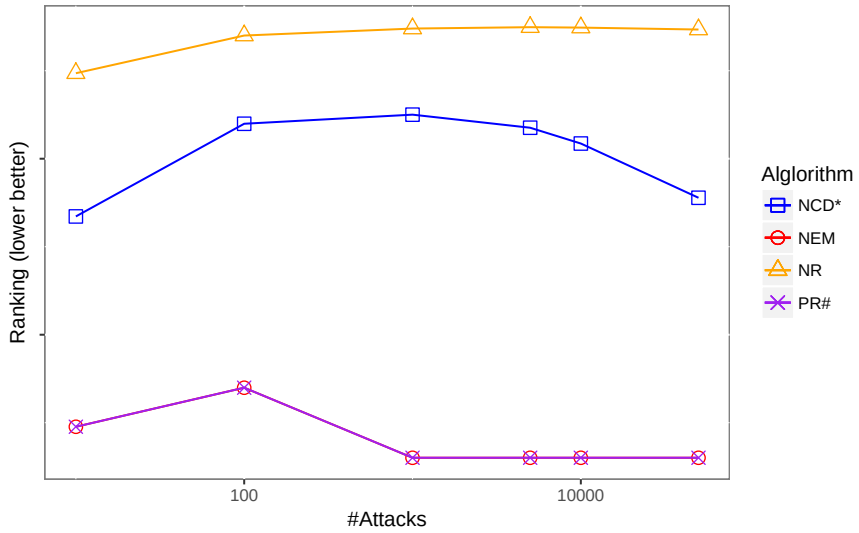


Fig. 4: Schematic of Loop Attack Utilizing Exchange Address The first and h-th loop attacks are shown in the figure. The selected authoritative Exchange node is Shapeshift; Edge label indicates temporal sequence; The transfer amount between nodes controlled by Sybil attacker and Shapeshift node is YEther; There are X loop transfers during the manipulation.

We choose Shapeshift(0x70faa28a6b8d6829a4b1e649d26ec9a2a39ba413) as the authoritative Exchange address. The results is shown in Fig.5: 1) As is shown in Fig.5a, with the attacker investing more capital, no algorithm is able to prevent the attacker's rank from being better, while the transaction graph defined in §2.2 reduces the attacking effect. **Nebulas Rank** can not only prefer nodes with high transaction throughput, but also resist manipulation to some extent; 2) As is shown in Fig.5b, with the attacker making more loop transfers, the transaction graph defined at §2.2 can let the attacker's rank be worse. The reason is that such transaction graph considers factors like coinage and encouragement value. Meanwhile, **Nebulas Rank** could strengthen these factors, bringing more resistance against manipulation.



(a) The effect of attacker's capital size on attacker's rank, with the number of loop transfers is fixed as $5,000$. (All axises are in logarithmic form)



(b) The effect of number of loop transfers on the attacker's rank, with attacker's capital fixed as Ξ5000 (All axises are in logarithmic form)

Fig. 5: Resistance against manipulation

The attacking method is shown as Fig.4; x-axis represents attacker's capital; y-axis represents attacker's ranking order (larger ranking order means the attacker fails to get better score, indicating higher resistance ability of the algorithm)

NR: The transaction graph is defined at §2.2, and ranking algorithm is described at §2.3;

NCD*: The transaction graph is defined at §2.2, with NCDawareRank algorithm;

NEM: The transaction graph is introduced by [38], with NCDawareRank

PR#: The transaction graph is introduced by [38], with PageRank algorithm

The damping factor of PageRank is 0.15; The clustering algorithm used by NCDawareRank is pscan[12], $\eta = 0.75$, $\mu = 0.1$

2.5 Related Works

Centrality, the core ranking index, is a most studied concept in network science since decades ago[39]. There are a body of literatures introducing various centralities, including degree centrality[21], eigenvector centrality[4], Katz centrality[27], closeness centrality[50], betweenness centrality[22][23][24][43][40], PageRank[9], HITS[29], SALSA[51], etc. Besides, there are some fundamental works trying to clearly classify and review these measurements by a unified framework[5][6][33]. When designing **Nebulas Rank**, before proper centrality is adopted, first we need to consider the property of graph. Blockchain transaction graph's scenario is most similar to the money exchange flow network mentioned in [5]. But the related algorithms mentioned by their work, such as flow betweenness centrality[24] and random-walk betweenness centrality (aka. current betweenness centrality)[40], are compute intensive and do not satisfy the property of "computable" with the large scale of Blockchain transaction graph.

Since Bitcoin[37] system released in 2009, researchers have done some statistical and empirical analysis on Bitcoin's transaction graph[49][26][41][2], and some use the transaction graph structure to discuss anonymity in Bitcoin[34][44][46][20][19]. After other cryptocurrencies emerged and become popular, transaction graph analysis is conducted for more blockchains[13][1]. **Nebulas Rank** adopts their transaction graph concept, i.e. Entity Graph in [56], with minor revisions. That is, each account, or set of accounts belonging to the same people, is mapped as a node, and each directed edge represents the intensity of transferring between two accounts. Actually before blockchain system like Bitcoin was invented, scientists have tried to study some financial networks among banks and global trading entities[48][8][52][3][17][36][7][30][53]. Comparing with blockchain transaction graph, these early studied finical networks are defined not only by transferring activities, but also by extra information such as loan. Moreover, the scale of these networks is much smaller. To conclude, there is rarely research work proposing custom ranking method for large scale transaction graph, especially blockchain transaction graph.

The most relevant work with **Nebulas Rank** is NEM[38]'s Proof-of-Importance scheme. It adopts NCDawareRank[42], which exploits the clustering effect of network topology, as the ranking algorithm, with clustering algorithm based on SCAN algorithm[57][54][12]. Although community structure does exist in transaction graph and should be helpful to handle with spam nodes, it does not guarantee that all nodes in Blockchain world controlled by one entity in the real world are mapped into one cluster, which leads to large room for manipulation. Besides, Fleder, Kester, and Pillai [20] uses PageRank[9][45] as an assisting metric to discover interesting addresses and analyze their activities. However, their work does not provide an automated framework to identify important nodes. Instead, it still relies on subjective analysis, which does not match **Nebulas Rank**'s context. The algorithm we choose is LeaderRank[14][31]. It is a simple yet effective variant of PageRank[9][45]. In PageRank, every node is assigned with identical teleportation parameter, while LeaderRank adds a ground node, assigning different teleportation parameters for each node. The weighing scheme of **Nebulas Rank** is partly from Li *et al.* [31]'s design, which allows nodes with more in-degree to be more likely visited by teleportation. Adopting LeaderRank algorithm could

yield results more suitable for the scenario of Blockchain.

3 Nebulas Force

We use Nebulas Force (NF) to describe the evolving capability of the blockchain system and its applications. As the first driving force of the blockchain system and its application development, the Nebulas Force includes three aspects, that is, the Nebulas Virtual Machine (NVM), the upgrade of the protocol code in the blockchain system, and the upgrade of the smart contract running on the blockchain system.

In Nebulas, we will introduce LLVM to implement the Nebulas Virtual Machine (NVM). The protocol code and the smart contract code will be compiled into NVM bytecode, which is dynamically compiled and optimized with the LLVM just-in-time (JIT) compilation function and eventually executed in the sandbox environment. Meanwhile, with the modular architecture of LLVM, developers can use their familiar programming languages to implement safer and higher-performance smart contracts, providing users with various decentralized applications.

For the upgrade of the protocol code in Nebulas, Nebulas will add the protocol code to block structure to carry out the upgrade of the protocol code by supplementing additional data on chains so as to avoid the possible split or bifurcation between developers and communities. With the development of Nebulas communities, the upgrade capability of NF and basic protocols will be gradually open to communities, and communities will define the evolving direction of the Nebulas and achieve its upgrade target. With the help of the core technology and the opening concept of NF, Nebulas will have a continuously evolving space and an infinitely evolving possibility. For example, a series of parameters including the NR algorithm parameter, the PoD incentive amount, the consensus algorithm and the production rate of new tokens can be gradually adjusted during the development of Nebulas without upgrading most client codes.

The smart contract is usually considered to be permanent and does not support upgrading. With the help of the design of underlying storage of the smart contract to support the cross-contract visit of state variables, Nebulas can complete the upgrade of the smart contract. This solution is very friendly to developers, making them respond to bugs more rapidly, which can prevent huge losses to users caused by any hacker events.

3.1 Nebulas Virtual Machine (NVM)

We will introduce LLVM [32] as the main component of NVM and LLVM bytecode as the NVM bytecode. The NVM bytecode is dynamically compiled and optimized through LLVM JIT and is executed in the NVM sandbox environment. With this architecture design, the performance and security of the core code and the smart contract of Nebulas can be continuously improved with the introduction of LLVM.

LLVM is a collection of highly modularized compiler toolchains and technologies, which was used as a code compilation framework in Google, Apple and many other companies. LLVM provides neutral intermediate representations (LLVM IR) and the corresponding compilation infrastructure, and offers a brand new set of compilation strategies regard to these infrastructure, including optimization of LLVM

IR, code generation from the LLVM IR to the LLVM bytecode and direct execution of the LLVM bytecode in different hardware platforms via the LLVM JIT, shown in Fig.6.

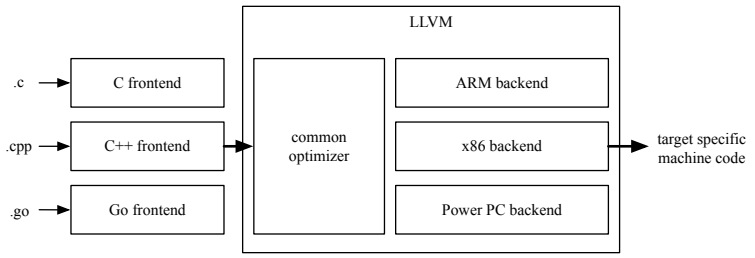


Fig. 6: LLVM

We construct the NVM based on LLVM, shown in Fig.7. First of all, we provide the underlying API libraries for blockchains. After that, we construct compiler frontend for different languages, such as Solidity, JavaScript, C/C ++, Go, etc. Then, we use the toolchain provided by LLVM to generate the LLVM bytecode. Finally, the LLVM bytecode is executed in a safe sandbox environment provided by NVM through JIT engine of LLVM.

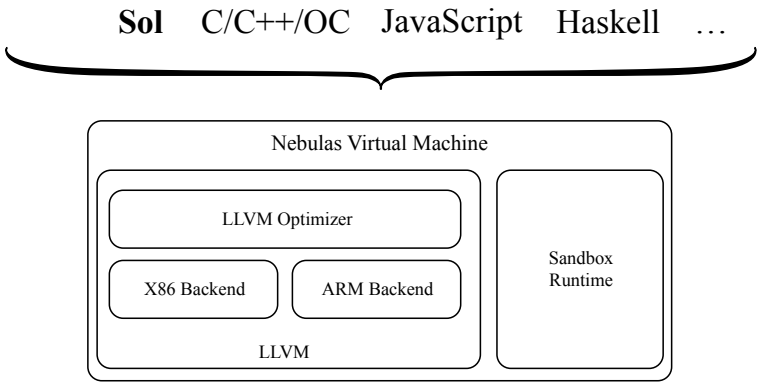


Fig. 7: Nebulas Virtual Machine

NVM is an important cornerstone of the Nebulas Force. When any new protocol code or smart contract is released, the LLVM bytecode is generated after the new code is complied by the LLVM compiler module in NVM and is released to the chain. After confirmed on chain, the new code will be complied and optimized by LLVM JIT, and then into the sandbox to replace the old code and be executed, shown in Fig.8.

With LLVM (see Fig.6), NVM also supports developers to develop smart contracts and applications with their familiar programming languages, such as Solidity, more flexible JavaScript, and even pure functions type of language Haskell. In addition to these popular languages, NVM can also provide customized high-level languages for different areas and scenarios, such as DSL (domain-specific language)

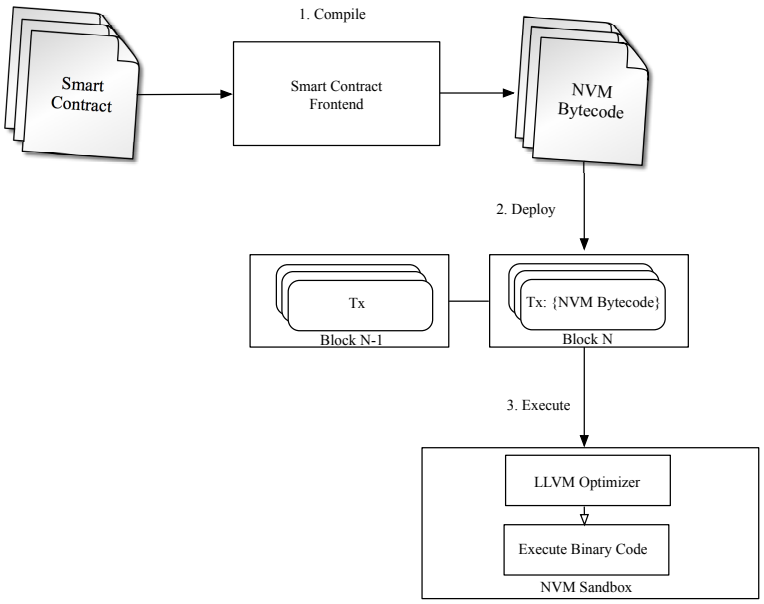


Fig. 8: The operation mechanism of the Nebulas Virtual Machine

for financial systems. These high-level languages are easier to be formally verified, further improving code robustness and security, and more conducive to the developers developing richer Smart contract and application.

3.2 Upgrade Design of the Protocol Code

We first give the block structure of the Nebulas, and then discuss how to upgrade the protocol code based on the block structure.

3.2.1 Block Structure

The Nebulas block data structure contains, but is not limited to, the following:

- Header：Block Header
  - Height：block height
  - ParentHash：parent block hash
  - Ts：timestamp
  - Miner：miner address
  - Dynasty：the consensus dynasty of the block
  - Epoch：the consensus age of the block
  - StateRoot：state root hash
  - TxsRoot：transaction root hash

  - ReceiptsRoot: transaction receipt hash
  - TransNum: number of transactions
- Transactions: transaction data (including multiple transactions)
  - From: transaction sender address
  - To: transaction receiver address, for creating a smart contract with a value 0x0
  - Value: transfer amount
  - Data: Transaction payload. It's the smart contract bytecode if the transaction is for creating a smart contract; It's the name of the calling function and the entry if the transaction is a smart contract call.
  - Signature: transaction signature
  - Gas: gas limit
  - GasPrice: gas unit price
  - Nonce: the uniqueness of transactions
- Votes: Prepare and Commit Votes (including multiple), used in PoD (see §5) consensus algorithm
  - From: voter
  - VoteHash: hash of the block voted for
  - Hv: the height of the block voted for
  - Hvs: the height of an ancestral block of the block voted for
  - VoteType: voting type, Prepare or Commit
  - Signature: vote signature
- Protocol Code: The protocol code (only 0 or 1 in a block)
  - Hash: hash of the protocol code
  - Code: the bytecode of the protocol code
  - ValidStartBlock: the start block number that protocol come into effect
  - Signature: signautre (sign by the developer community)
  - Version: the protocol code version number, and each upgrade needs to be incremented to prevent malicious accounts from rolling back to the old protocol code
  - Nonce: the uniqueness of protocol code
- Nebulas Rank: Nebula index (calculated once a week, most blocks haven't this section)
  - RankVersion: NR version
  - RankRoot: NR rank hash
  - RankRecords: NR rank record
    * Address: Account address tag

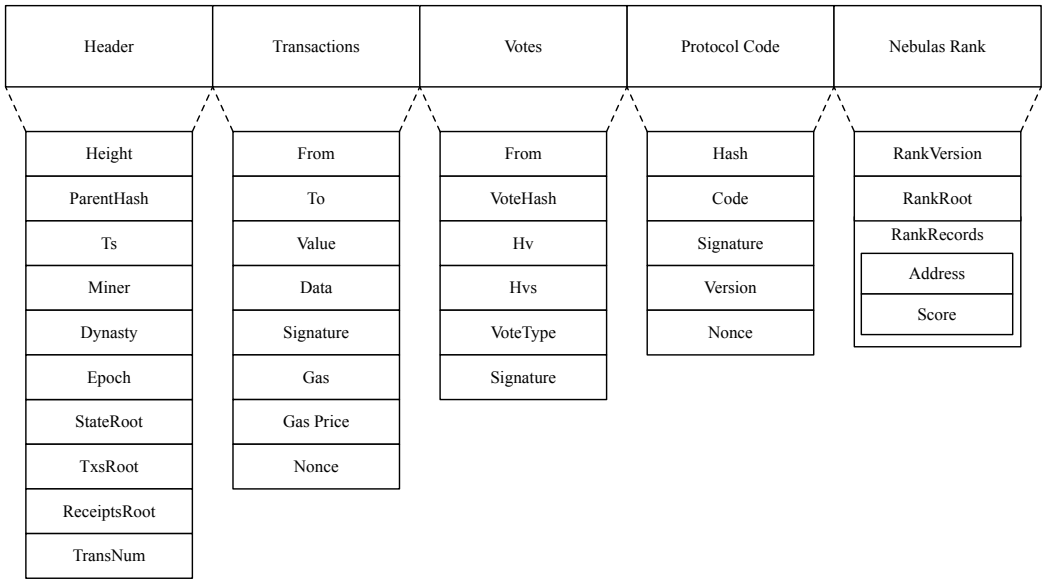


Fig. 9: Block Structure

* Score：NR value

Similar to other cryptocurrency systems, the interaction between the account and the blockchain is done through a particular transaction. The account creates a transaction, which is signed with a private key, and sends it to any node in the blockchain and broadcasts it to full network node through the P2P network. During the fixed block time interval, the nodes specified by the PoD consensus algorithm (see §5) collect all transactions in the time and pack them into blocks of standard format and broadcast them to the rest of the network. After verification by each node, the new block is appended into the local ledger and becomes a part of the global ledger.

In Ethereum, transactions are divided into two types: ordinary account transaction and smart contract transaction. We add new transaction types to the blocks of Nebulas: protocol code and the Nebulas Rank. The protocol code, as a part of the blockchain data, is stored on the chain, and the upgrade of the basic protocol of Nebulas is carried out through supplementing additional data on chains. Based on the NR ranking algorithm, the NR value of each account is calculated in each cycle and saved in the corresponding chain to facilitate the real-time NR value calling and historical ranking query.

3.2.2 Upgrade of the Protocol Code

The Nebulas client node can obtain the compiled virtual machine bytecode (NVM bytecode) from the storage area of the Protocol Code in the latest block. If there are no data of the Protocol Code in the latest block, it indicates that the protocol code has no change and has to trace back to the Protocol Code in the nearest block. Any actions of the protocol code of blockchains will be determined by the Protocol Code, including the authentication algorithm, the packing rules, the NR algorithm, the incentive

mechanism, etc. Almost all actions of blockchains can be defined by the Protocol Code.

If the protocol code needs to be upgraded, the Nebulas development team will be responsible for its development and the code will be released to open channels for discussion and voting in communities. Voting can be carried out in the form of the smart contract or voting in the forum. If the majority of community members agree to upgrade the protocol, the Nebulas development team will pack the latest code into Protocol Code transaction, and release it to all nodes of the whole network; only if the bookkeeping nodes include it in blocks, it will become valid at the specified block height. This type of blockchain protocol upgrade is transparent to clients without soft and hard fork.

In order to ensure that the protocol code is released after authorization, the publisher of the Protocol Code uses the address reserved by the core Nebulas, cannot be changed with hard code within the genesis block. All bookkeeping nodes will verify the signature of Protocol Code. If the signature fails to pass the verification, it will be deemed as the illegal data.

The subsequent improvement measure is to change the signature verification of the Protocol Code into the M-of-N multi-signature, which can also be implemented through the upgrade of the Protocol Code.

3.3 Upgrade Design of the Smart Contract

3.3.1 Turing Complete Smart Contract Programming Language

A smart contract is a set of promises defined in the digital form, including agreements for the execution of those promises by the contract participants. Physically, the carrier of the smart contract is the computer code that the computer can recognize and can be operated on the computer. Bitcoin script language is an imperative and stack-based programming language, and because it is Turing incomplete, there are some limitations on its applications. The Ethereum is the world's first blockchain system that implements Turing complete smart contracts. The adopted programming language is Solidity and Serpent, enabling developers to develop a wide variety of applications in a rapid and efficient manner. After the smart contract code is published on the blockchain, it can be automatically executed without participation of any agencies.

In the early stage, the smart contract programming language in the Nebulas was fully compatible with Solidity of Ethereum, making it easy for developers to migrate the smart contract applications developed for Ethereum into Nebulas seamlessly. We add some instruction sets related to the Nebulas Rank to Solidity language, facilitating developers to obtain the NR value of any users. After that, based on NVM, we provide support to various programming languages, making it easy for developers to program with their favorite advanced languages, such as Java, Python, Go, JavaScript, Scala, etc., or even create customized applications with certain advanced languages in a specific field.

3.3.2 Upgradable Contracts

Currently, for the design of the smart contract of Ethereum, once the code is made, it cannot be changed any more. From the moment that the code logic is made, it cannot be upgraded forever. If the smart contract serves as an agreement, it is required to be unchangeable, which represents an agreement that its operation is determined. However, as the smart contract has been increasingly widely used, its workflow and code become more and more complicated. It is found that it looks like a real contract in the real world. If it fails to be reviewed carefully, it is difficult to prevent human errors in the design and coding process. Once hackers find any bugs, the loss is always very significant. In June 2016, the DAO Attack was due to a code defect, causing a total loss of $60 million to the users of Ethereum. In addition, a recent bug of Parity Wallet resulted in a loss of 150,000 ETH, valued at $30 million. Since the bitcoin is designed with Turing incompleteness and many script instructions have been deleted, its security level is very high.

Although there are many best practices in smart contract programing, more strict reviewing process and even formal verification tools to verify the certainty of the smart contract through mathematical proofs, codes cannot be totally bug-free. When looking back into our currently centralized Internet world, we find that Internet services can be upgraded so as to fix different bugs emerged in the development process. Perfect application system cannot be designed, but is evolved gradually. We believe that the fundamental requirement of sorting out security problem of the smart contract is to formulate an upgradable design solution for the smart contract.

There are some solutions to the upgradeable design of the smart contract in Ethereum, which are generally divided into two categories. The first one is the Proxy Contract available to the public. Its code is very simple, only forwarding the request to the following real function contract. When the contract needs upgrading, just make the pointer of the internal function contract of the Proxy Contract point to the new contract. The second one is to separate the code contract from storage contract. The storage contract is responsible for providing the external contract with methods to read and write the internal state. The code contract is responsible for the real business logic, and when upgrading, it is only responsible for deploying the new code contract so as to lose no state. These two solutions have their own limitations correspondingly, so they cannot solve all the problems. For example, the separation between the code contract and storage contract makes it more complicated. Sometimes it is even unfeasible. Although the Proxy Contract is able to point to the new contract, the state data of the old contract cannot be migrated. Some contracts are not well designed at the initial development stage, so they fail to leave any interface for later upgrade.

We designed a simple smart contract upgrade solution. In this way, at the language level, another contract can read and write a contract state variable directly (in line with the security constraints). For example, there is a Token contract, and its code is as shown in Table 10.

When the contract is deployed, the balances variable is marked with the keyword of shared, and when

```
contract Token {
  mapping (address => uint256) balances shared;

  function transfer(address _to, uint256 _value) returns (bool success) {
     if (balances[msg.sender] >= _value) {
       balances[msg.sender] -= _value;
       balances[_to] += _value;
       return true;
     } else {
       return false;
     }
   }
   function balanceOf(address _owner) constant returns (uint256 balance) {
       return balances[_owner];
   }
}
```

Fig. 10: Original Contract Code

it is compiled into bytecode for operation, the virtual machine will design the storage area separately for this variable. For the variable that is not marked with the keyword of shared, it cannot be accessed directly by other contracts.

If a bug in the transfer function of the original code needs modifying, check the _value and deploy the new smart contract code as shown in Table 11.

After the new contract is deployed, the old contract can choose selfdestruct, which means that it cannot be accessed any more. But the shared variable will be permanently reserved. The new contract can completely inherit the asset of balances from the old contract without losing any state, so no additional migration is required. However, when developing a smart contract, it is necessary to declare the critical state variable to be shared. The compiler will specially handle the storage area of the variable so as to ensure that it can be accessed by other authorized contracts.

In order to ensure the security, the contract upgrade and the old contract must use the same creator, otherwise an exception will be thrown during operation.

There is a moral problem in this design. Once the provisions of the contract has been proposed and concluded, they should not be modified. Any change later should be agreed by the contract audiences. We plan to introduce a voting mechanism to approve the upgrade of the smart contract, preventing the contract from being modified silently by the contract creator.

```
[baseContractAddress="0x5d65d971895edc438f465c17db6992698a52318d"]
//baseContractAddress is the address of the old contract
contract Token {
  mapping (address => uint256) balances shared;

  function transfer(address _to, uint256 _value) returns (bool success) {
     if (balances[msg.sender] >= _value && _value > 0) {
       balances[msg.sender] -= _value;
       balances[_to] += _value;
       return true;
     } else {
       return false;
     }
   }
   function balanceOf(address _owner) constant returns (uint256 balance) {
       return balances[_owner];
   }
}
```

Fig. 11: New Contract Code

With this upgradable solution, the DAO Attack or Parity bug or similar bug attack events can be fixed more rapidly, rather than through hard fork. After repair, assets of all users can continue to be used without migration.

4 Developer Incentive Protocol (DIP)

4.1 Design Goals

In order to create a good community for Nebulas, we propose the Developer Incentive Protocol (DIP) for smart contract developers and express our appreciation to outstanding smart contract developers who contribute to Nebulas by rewarding them NAS.

4.2 DIP Reward Distribution Algorithm

We believe that an excellent smart contract depends on how many users are willing to use it. More high-value accounts, better smart contract. As a kind of universal value criteria of accounts, NR can be used to assess high-value accounts. The design of DIP combines NR and the common WAU (weekly active users) concept, the total WAU value measure is used to evaluate the value measure of the smart contract.

DIP is carried out once a week. For Smart Contract C, it is assumed that the active account address set of this week is WAA (Weekly Active Addresses). According to the NR ranking in §2.3 (Top X is taken), the sum of NRs of weekly active addresses is calculated as the SCS (Smart Contract Score) of Contract C, as shown in Equation 9.

$$SCS(C) = \sum_{addr \in WAA} (max\{X + 1 - NR(addr), 0\}) \tag{9}$$

Based on weekly SCSs, they are sorted out from high to low for SCR (Smart Contract Rank). Top N smart contracts are taken, and the corresponding developers will share M NASs based on proportion. In order to avoid malicious ranking scam, the DIP distribution curve is designed to be even as shown in Fig.12, but it is still ensured that the revenue of Rank 1 is 2 times of that of Rank N to indicate the difference in SCS. The proportion constraints are as shown in Equation 10.

$$\begin{aligned} Coin(C) = \quad & kln(N + 1 - SCR(C)) + b \\ \text{s.t.} \quad & kln(N) + b = 2b \\ & \sum_{x=1}^{N} (kln(x) + b) = M \end{aligned} \tag{10}$$

DIP rewards will be calculated separately and distributed by each node. Assuming that one block in Nebulas is generated every S second (s), DIP rewards will be calculated once every 24*7*3600/S blocks for all nodes, and will be distributed to the corresponding smart contract cash-out addresses.

In order to encourage the diversity of Nebulas ecological smart contracts and stimulate outstanding results of more new developers, DIP stipulates that each smart contract can be rewarded up to K time(s).

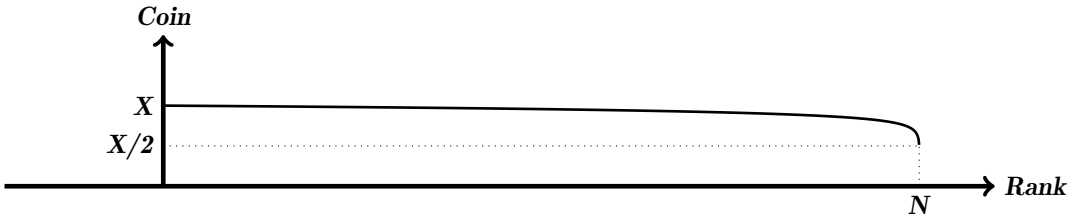


Fig. 12: DIP Reward Distribution Curve

DIP will select Top N smart contracts qualified for rewards according to ranking so as to promote the development of the ecology construction of blockchain applications.

4.3 Experimental Results

We collected the transaction data occurred in May 2017 from Ethereum and calculated the DIP ranking of the first week, as shown in Table 2.

Table 2: Top 10 of DIP Ranking Results of the First Week in May 2017[1]

Contract Address	Score	Description[2]
0xa74476443119a942de498590fe1f2454d7d4ac0d	264456363.0	GolemToken
0x49edf201c1e139282643d5e7c6fb0c7219ad1db7	207900181.0	TokenCard-ICO
0x48c80f1f4d53d5951e5d5438b54cba84f29f32a5	129625776.0	REP-Augur-OLD
0x6810e776880c02933d47db1b9fc05908e5386b96	108324489.0	Gnosis-TokenContract
0x6090a6e47849629b7245dfa1ca21d94cd15878ef	54429341.0	ENS-Registrar
0x607f4c5bb672230e8672085532f7e901544a7375	48526808.0	RLC
0x8d12a197cb00d4747a1fe03395095ce2a5cc6819	46498412.0	etherdelta_2
0xf7b098298f7c69fc14610bf71d5e02c60792894c	43746158.0	GUPToken
0xaaaf91d9b90df800df4f55c205fd6989c977e73a	42026627.0	TokenCardContract
0xaec2e87e0a235266d9c5adc9deb4b2e29b54d009	41427314.0	singularDTVToken

[1] block range [3629091, 3665815]

[2] from etherscan.io

It can be seen that the top-ranked contracts are more famous and more active in the calculation cycle, which are in line with our original intention of motivating the eco-builders.

4.4 Cheating Analysis

The smart contract can only be called passively. Therefore, if a cheater wants to increase his/her smart contract ranking, he/she has to find sufficient high-NR ranking accounts to call his/her contract.

First of all, it will be impossible for cheaters to improve their DIP ranking at the cost of zero. Provided that a cheater wants to improve his/her ranking of Contract C and forges a great number of accounts for it. However, when SCS is calculated in Equation 9, only Top X of SCS of calling in the NR ranking is greater than 0, while the NR ranking of his/her newly forged accounts will be outside Top X. Even if Contract C is called, it will impose no impact on the DIP ranking.

Secondly, if a cheater is willing to pay a certain price for improving his/her DIP ranking of the contract, two options are available for him/her. The first one is to forge accounts with high NR and call Contract C at his/her cost so as to improve the ranking of Contract C. It has been analyzed for forging accounts with the high NR in §2.4. To improve the NR for each account, a vast sum of money is required to forge the special topology of such account. Besides, as NR is updated periodically, it will be very expensive to maintain high NR. The second one is to find a great number of accounts with high NR and persuade or bribe their owners to call Contract C. However, it is difficult for such off-chain actions to be scaled up. What's worse, these accounts with high NR found by the cheater at a great cost will account for only a small part of Top X, which will impose almost no impact on really outstanding contracts.

5 Proof of Devotion (PoD) Consensus Algorithm

5.1 Design Goals

The consensus algorithm is one of the cornerstones of blockchains, and its rapidity and irreversibility are our focus. In addition, in order to build a good ecology of Nebulas, we believe that fairness is equally important. If the big capital can easily gain power to control the block consensus in Nebulas, the interests of many developers and users will be damaged. It is difficult for an ecology which cannot guarantee the interests of contributors to create in-depth value, as it goes against the design principles of Nebulas. Thus, the consensus algorithm should be designed to ensure the rapidity and irreversibility first, on which basis we will pursue fairness as much as possible, so as to guarantee the interests of contributors in Nebulas.

5.2 Defects of Commonly Used Consensus Algorithms

We have tried to find suitable commonly used consensus algorithms that match with our design goals, but these algorithms cannot completely meet our requirements.

The PoW (Proof of Work) consensus algorithm is a zero-sum game, which uses the competitive hash calculation to determine the bookkeeper, rendering a great amount of electric power in the whole ecology wasted in the competition when any blocks are fed out, and thus the mining cost is high and the speed is restricted. With the increase of nodes involved in mining, the probability of each node to obtain the bookkeeping right will be reduced, leading to a continuous rise in the cost of stable feed-out of blocks under the PoW protocol. The Bitcoin, which continues to increase the difficulty of mining, has to face the situation sooner or later where the mining machine cannot make ends meet, while Ethereum has long been considering the use of the new PoS consensus algorithm Casper [55] to gradually replace the current PoW consensus [11]. It can be seen that in view of the mining speed and the economic cost, the PoW is not beneficial to the long-term rapid development of the ecology of Nebulas, which is against our "rapid" goals.

The PoS (Proof of Stake) consensus algorithm attempts to use the asset amount to replace the hash rate and distribute the probability of obtaining the bookkeeping right according to the coin age or deposit amount. Currently, both Peercoin [28] and the Casper Protocol of Ethereum adopt the PoS consensus algorithm. This algorithm overcomes the shortcoming of the high power consumption of PoW but visually enlarges the impact of the capital on the probability distribution of the bookkeeping right. Compared with PoW, the big capital under PoS is more likely to gain power to control the ecology and form large group monopoly, possibly damaging the interests of the ecology contributors and impacting adversely on the value generation of Nebulas, all of which are against our "fairness" goal.

The PoI (Proof of Importance) consensus algorithm was first proposed by Nem [38]. Different from PoS, the concept of account importance is introduced to PoI, and the account importance score is used

to distribute the probability of the bookkeeping right. This algorithm overcomes the shortcoming of the high power consumption of PoW and relieves the PoS capital monopoly crisis, but exposes the "nothing-at-stake" problem. The cost for a cheater to reverse a block is significantly reduced, which goes against our "irreversibility" goal.

In short, in view of the discrepancy between commonly used consensus algorithms and our goals, we have proposed the PoD (Proof of Devotion) algorithm to integrate PoI, which evaluates the comprehensive account influence, with PoS, which involves strict economic penalties. PoS enhances the irreversibility of PoI, while PoI reversely contains the monopoly of PoS, which facilitates the free and rapid development of the ecology.

5.3 Design of the PoD Algorithm

5.3.1 Generation of New Blocks

Similar to the PoI consensus algorithm that selects highly important accounts, the PoD selects the accounts with high influence in the ecology. The difference lies in that the PoD empowers the selected accounts to have the bookkeeping right with equal probability to participate in new block generation in order to prevent tilted probability that may bring about monopoly.

When selecting accounts with high influence, we use NR, the universal measure of value generated from Nebulas. In the algorithm design of NR, we highlighted the liquidity and propagation of accounts (see §2.1). We believe that the accounts featured with these properties have a high influence with regard to the ecology construction. Thus, in the PoD, the accounts ranked Top N in the NR will be selected, and after these accounts voluntarily pay a certain number of NASs as the deposit, they will be qualified as the validator of new blocks to participate in bookkeeping.

After the validator set is provided, the PoD algorithm uses the pseudo-random number to determine which one in the set is the new block proposer, which need to pack recent transactions to generate the new block. The validator set is changeable. The eligible account can choose to join or quit the set. Besides, the eligible accounts may vary with the periodical change of NR. Therefore, we designed the dynamic validator set change mechanism in the PoD to implement the change of the validator set.

5.3.2 Dynamic Validators Set

The validator set changes the same way as a dynasty, so the set is divided into different dynasties, and the validator set within a dynasty will not change. A dynasty cannot experience too rapid change, and no change should be made within a period of time. Thus, we define every X blocks as an Epoch, and in the same Epoch, no change takes place in the dynasty. Therefore, the change of dynasties will only occur when one Epoch is handed over to another. At that time, the first block of the previous Epoch will be investigated. If this block reaches the finality state, then the current Epoch will enter into the next dynasty of D1; otherwise, the previous dynasty of D0 will remain; the process of which is shown in

Fig.13.

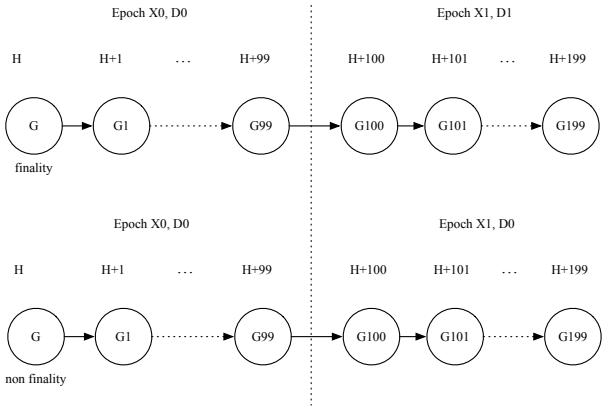


Fig. 13: Change of validator dynasties (assuming X = 100)

Because of the network delay, the finality state of Block G at each node may not be the same when any change of dynasties takes place. Therefore, by reference to dynamic Casper validator set strategies, it is required that the consensus process of each dynasty will be completed jointly by the validator sets of the current and previous dynasties. Therefore, in any dynasty, an eligible account can only apply for joining or quiting the validator set of Dynasty D+2, and when the dynasty evolves into Dynasty D+2, it can participate in the consensus process of the new block.

5.3.3 Consensus Process

After a new block is proposed, all the people in the validator set of the current dynasty will participate in a round of BFT-Style (Byzantine-Fault-Tolerant Style) voting to determine the legitimacy of this block. In the beginning of voting, each validator who participates in this block consensus will be charged 2x (x is the incentive bonus proportion) as the deposit and then the two-stage voting process will be kicked off.

- **In the first stage**, it is required that all validators vote *Prepare* tickets for the new block. After voting the *Prepare* ticket, the validator will be rewarded $1.5x$ bonus. If the validators holding over two-thirds of the total deposits in both current and previous dynasties vote *Prepare* tickets for the new block, this block will enter into the second stage of voting. It should be noted that the proposer of the new block votes the *Prepare* ticket for the new block by default.
- **In the second stage**, it is required that all validators vote *Commit* tickets for the new block. After voting the *Commit* ticket, the validator will be rewarded $1.5x$ bonus again. If the validators holding over two-thirds of the total deposits in both current and previous dynasties vote *Commit* tickets for the new block, this block will reach the finality state.

In order to speed up the development of the entire ecology, if the difference between the timestamp

of *Prepare* ticket and *Commit* ticket in Block b and the timestamp of Block b exceeds T, then these tickets will be considered expired and will be ignored directly.

5.3.4 Fork Choice

The PoD algorithm selects the canonical chain according to the block score at each height. It always selects the block with the highest score to join the canonical chain, and the score of Block b at Height h is as follows:

$$Score(b,h) = \sum_{(b',h') \in children(b)} Score(b',h') + \sum committed\ deposits\ in\ b \tag{11}$$

Namely the sum of deposits corresponding to *Commit* tickets received by this block and all of its descendant blocks, as shown in Fig.14.

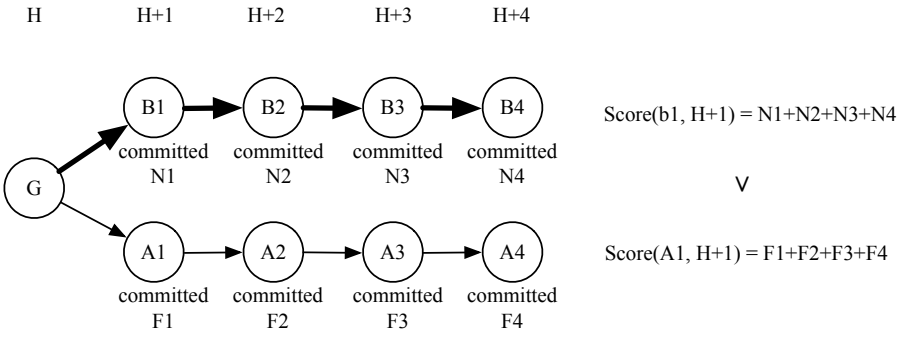


Fig. 14: Fork Choice Example

5.3.5 Slashing Conditions

To avoid any malicious damage to the consensus process, which may result in failure in the completion of consensus process and obstruction of eco-development, PoD constrains consensus activities of validators based on reference to Casper's minimum slashing conditions [35].

Assume that *Prepare* vote and *Commit* vote in the consensus process have the following structures:

- $Prepare(H, v, vs)$, where H is the hash value of the current block; v represents the height of the current block; vs represents the height of a certain ancestral block of v.
- $Commit(H, v)$, where H is the hash value of the current block; v represents the height of the current block.

PoD algorithm defines the following 4 basic rules for the entire voting process:

- There is a strict order in the two-stage consensus process of a single block: Only when the total deposits of $Prepare(H, v, vs)$ votes for the first stage reaches 2/3, can validators deliver $Commit(H, v)$ votes for the second stage.
- For multiple blocks, there is no mandatory rule that only when the consensus process of one block is finished, may consensus process of the next block begin. Interwoven consensus is permitted providing that it is conducted based on a certain order. Only after the first stage of process for height vs is finished and the proportion of $Prepare(H, vs, vs')$ votes reaches 2/3, can $Prepare(H, v, vs)$ votes for its descendant blocks be delivered based on vs in order to ensure stable proceeding of the interwoven consensus.
- To avoid the conduct of malicious cross-block voting of any node by taking advantage of the interwoven consensus, it is required that after the delivery of Prepare (H, w, u) votes based on the height of u, no Commit(H, v) vote can be delivered in all blocks with a height within the range from u to w, thus guaranteeing high efficiency and orderliness of the consensus process.
- For the purpose of preventing nodes from staking with one deposit on multiple branches simultaneously, which may lead to the problem of "nothing-at-stake", it is required that after $Prepare(H1, v, vs1)$ votes are delivered at a certain height, no different $Prepare(H2, v, vs2)$ vote can be delivered again.

Once being reported and verified, any validators breaking the above rules will be punished and all his/her deposit will be confiscated, in which 4% will be shared by whistleblowers as a reward and the remaining part will be destroyed.

5.4 PoD Economic Analysis

5.4.1 Incentive Analysis

A validator participating in the PoD algorithm will be rewarded with $1x$ NAS on each legal block. In case of failure in finishing the *Prepare* stage and entering into the *Commit* stage due to poor network traffic or any cheating behavior, the validator will lose $0.5x$ NAS. Therefore, any validator becoming value nodes will secure a large amount of earnings from accounting under good network traffic when not engaging in any cheating behavior.

5.4.2 Cheating Analysis

Double-spend Attack

If it is assumed that a merchant confirms transaction and makes delivery when the new block reaches the status of finality, then the minimum cost to be paid by a fraud for realizing zero-cost shopping through completion of double-spend attack under the PoD consensus algorithm is described as follows:

Firstly, the fraud needs to increase his/her Nebulas Rank to Top N, become a validator by paying a certain amount of NAS as deposit and apply for participation in validation of blocks in the D+2 dynasty.

Then, the fraud needs to be selected as the proposer of a new block by the pseudo-random algorithm. At this moment, the fraud proposes two new blocks at the same height, of which one block has a hash value of hash1 and contains a transferring transaction from the fraud to the merchant, while the other block has a hash value of hash2 and contains a transferring transaction from the fraud to himself/herself.

Finally, in order to make both of hash1 and hash2 blocks reach finality, as shown in Fig.15, the fraud has to spend 1/3 of the total deposits in this dynasty to bribe 1/3 of the validators and make them to deliver *Commit* votes to both blocks.

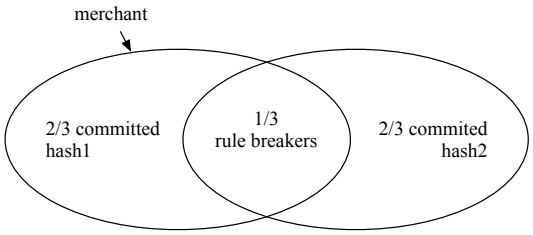


Fig. 15: Financial Punishment on Double Spend

Therefore, in order to complete a successful double-spend attack, the fraud needs to spend a certain amount of energy and financial resource to increase his/her Nebulas Rank (see §2.4 Resistance to Manipulation) and then spend at least 1/3 of the total deposits in current dynasty to make both of the blocks reach the finality status after he/she is luckily selected as a proposer.

51% attack

In PoW, launch of a 51% attack requires 51% of hashrate. In PoS, launch of a 51% attack requires 51% of deposit. However, in PoD, launch of a 51% attack requires 51% of accounts in the validators set, which means that a sufficient number of highly-reputed users need to rank at Top N in the Nebulas Rank and payment of a corresponding amount of deposit is required, so it will be more difficult to launch a 51% attack in PoD.

Short-range Attack

In PoD, blocks at each height have term of expiration time of consensus. Therefore, it is almost impossible to complete a long-range attack in PoD, but it is still possible to launch short-range attacks within term of expiration time.

When a short-range attacker (Attacker) attempts to forge A-chain to replace B-chain to become the canonical fork when blocks at the height of H+1 are still within term of expiration time, Attacker needs to ensure that score of block A1 is higher than that of block B1. Multiple voting will be severely punished, so it will be unavoidable for Attacker to bribe validators; otherwise, it is impossible to complete a short-range attack. For the purpose of presenting the safety of the PoD consensus algorithm, the costs to be paid by Attacker in reverting different numbers of blocks are analyzed as follows.

If Attacker plans to revert B1, the minimum cost to be paid by Attacker is as described in Fig.16, which is equivalent to a double-spend attack. If Attacker becomes the proposer of blocks at the height of H+1, then he/she has to bribe 1/3 of the validators in Dynasty D0 and make them conduct multiple voting in order to make A1 reach finality, for which the minimum cost is 1/3 of the total deposits.

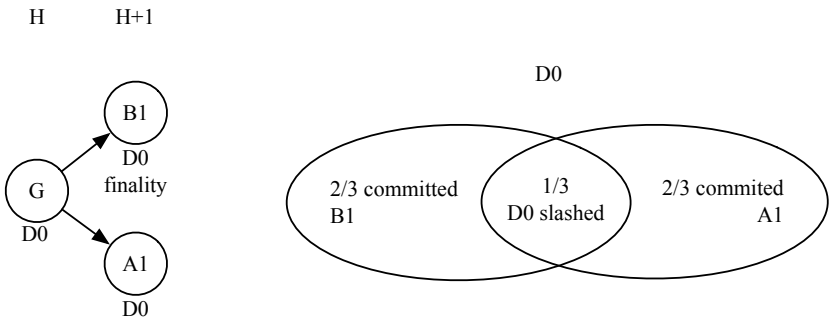


Fig. 16: Revert One Block by Short-range Attack

Assume that B1 and B2 have reached the status of finality and transactions in the blocks have come into effect. If Attacker wants to revert B1-B2, the following two circumstances are taken into consideration.

- The first circumstance is shown in Figure 17 (a): When Heights H+1 and H+2 are in the same Epoch and dynasty, Attacker needs to bribe 1/3 of the validators in D0 in order to make A1 reach finality. Meanwhile, these 1/3 of the validators will be punished and their deposits will be totally confiscated. During validation of A2, sum of deposits equals to 2/3 of deposits in A1. At this moment, if Attacker wants to secure the same amount of commit votes as B2, he/she has to bribe all the remaining validators without cheating and lose at least 3/3 of the total deposits. Even if Attacker succeeds in doing this, it is impossible to guarantee that score of A1 is higher than that of B1 and Attacker will face a high risk of failure of attack.
- The second circumstance is shown in Figure 17 (b): When Heights H+1 and H+2 are in different Epochs and different dynasties, Attacker needs to bribe 1/3 of the validators in D0 to make A1 reach finality and then bribe 1/3 of the validators in D1 to make A2 reach finality, so Attacker will lose at least 2/3 of the total deposits in order to complete such an attack. To sum up, to launch a short-range attack to cause invalidation of two blocks that have reached finality, Attacker needs to pay at least 2/3 of the total deposits.

If Attacker wants to revert B1-B3, as shown in Figure 18, Attacker needs to firstly bribe 1/3 of the validators in D0 in order to realize finality of A1 and then bribe 1/3 of the validators in D1 in order to realize finality of A2. Finally, Attacker needs to bribe all of the remaining 2/3 of the validators in D1 in order to realize finality of A3. To sum up, 4/3 of the total amount of deposits will be lost. It will be very difficult to prepare for these attacks. Even if an attacker manages to succeed in making necessary

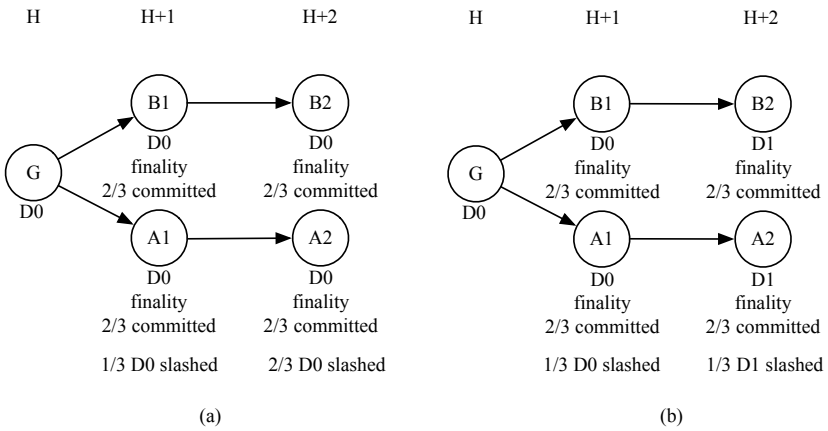


Fig. 17: Revert Two Blocks by Short-range Attack

preparations, he/she can't guarantee that score of A1 is higher than that of B1. Therefore, it is possible that such attack may fail.

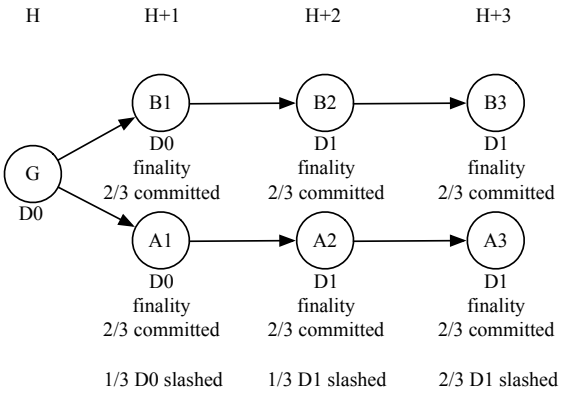


Fig. 18: Revert Three Blocks by Short-range Attack

If Attacker wants to revert B1-BN, in which N is limited by term of expiration time of consensus of the block and thus can't be a very large number. When $N = 3$, the total deposits of all validators in the current dynasty will be totally confiscated. Therefore, when $N >= 4$, it is impossible to complete an attack in order to make score of B1 higher than that of A1 and revert B1-BN. It is pointless to launch such an attack.

6 Blockchain Search Engine

6.1 Introduction

As an increasing number of smart contracts are deployed by developers, searching needs for massive smart contracts soar. Given that smart contracts are simply code and include no functional descriptions, indexing smart contracts with search engine technologies imposes a high difficulty. To index smart contracts properly, we use the following methods:

- Crawl webpage data relevant to smart contracts to set up mappings between the data and blockchain smart contracts.
- Encourage developers to upload verified source code of smart contracts, analyze the functions and semantics of the code, create indexes for the source code, and provide the searching function for similar contracts. For smart contracts without source code, decompile them for their source code.
- Establish standards for smart contracts so that any contracts matching those standards can be retrieved and found by users. Also, encourage developers to provide informational descriptions of contracts during smart contract creation.

```
contract SearchableContract {
    string public language;
    string public author;
    string public name;
    string public title;
    string public description;
    string public tags;
}
```

6.2 Infrastructure

In current stage, We think that the centralized search engine is more suitable for obtaining the best user experience and presenting the value of Nebulas Rank. The Nebulas development team is dedicated to developing a searching service, retrieving all smart contracts in real time, performing multilingual word breaking and creating full-text indexes to provide users with a user-friendly web interface. The impartiality of the NR ranking algorithm and the verifiability of each node ensure the impartiality of the centralized searching service, while the complete code of the searching backend is available to the

community. Also, third-party developers can create their own searching services on this basis. Fig.19 shows the architecture of the searching service.

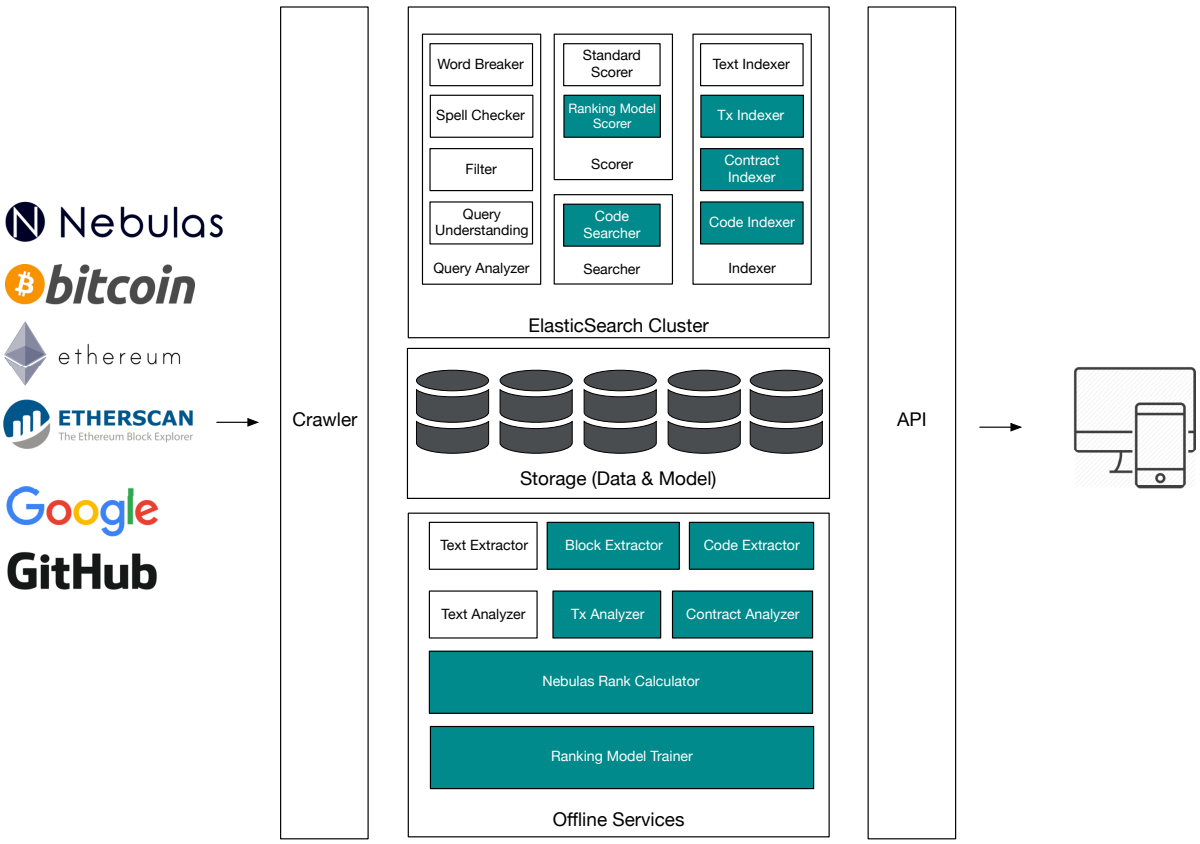


Fig. 19: Architecture of the searching service

- **Crawler** data sources of the Crawler in blockchain search engine are classified into two types: one for collecting block information and code of smart contracts from blockchains, while the other for crawling data about smart contracts from public URLs including introductions, Dapp user comments and news.
- **Extractor** consists of Text Extractor, Block Extractor and Code Extractor, which provide text information, block information and code extraction service for smart contracts, respectively.
- **Analyzer** consists of Text Analyzer, Tx Analyzer and Contract Analyzer, which are text information, block transaction information and smart contract analyzers, respectively. For the smart contract analyzer, it provides contract decompilation, source code extraction, semantic analysis and so on.
- **Nebulas Rank Calculator** refers to the Nebulas Rank calculator service, which is used to calculate the Nebulas Ranks of each non-contract and contract accounts offline.
- **Ranking Model Trainer** refers to the ranking model trainer service. Ranking rules take multiple factors into account: matching field, text relevance, NR Rank value of the contract, transaction

quantity of the contract, frequency and depth, NR Rank of the user conducting a transaction with the contract, and contract security. Based on users' actual use conditions, the machine learning algorithm (GBDT and artificial neural network (ANN) are optional) is used to train the ranking and scoring model, which is also constantly improved according to user feedback. The trained model is used by Scorer of the searching service.

- **Query Analyzer** refers to the keyword analysis service, which includes the multilingual word breaker (Word Breaker) and the spell checker (Spell Checker).
- **Indexer** creates proper indexes from Analyzer and supports both full and incremental indexing.
- **Scorer** is classified into two levels: Level-1 Standard Scorer recalls candidate result sets from ElasticSearch, which is done to recall as many candidate results as possible through the efficient and effective ranking in the ElasticSearch cluster. Level-1 can recall several thousands of results. Level-2 Ranking Model Scorer uses the offline rank model to calculate and reorder the rank of each Level-1 candidate result set. For this level, the calculated results feature a sounding accuracy and can be used directly by users.
- **Searcher** is responsible for communicating with the ElasticSearch cluster and packing and returning the search result to the searching frontend.
- **API** provides external applications with comprehensive searching API services.
- **ElasticSearch Cluster** ES refers to the server cluster. The Nebulas development team plans to use the open-source search engine ElasticSearch to support full-text indexing.

6.3 Nebulas Trends

The Nebulas creates the tendency list in combination with Nebulas Rank to provide user visual multi-dimensional values in blockchains.

- **Nebulas Rank list for non-contract accounts.** It displays the daily NR list and the NR quick rise and drop lists. Also, this rank list visualizes the rank variation tendency of each account and the health change tendency of the entire network.
- **Nebulas Rank list for contract accounts.** Based on the NR values of non-contract accounts, the Nebulas Rank list calculates the NR list of contract accounts, the quick rise and drop lists, the variation tendency of each contract and the tendency chart for the quantity and use frequency of smart contracts on the entire network. In addition, we will present other smart contract lists such as the token contract list and the market contract estimation list to display a wider dimension of information.
- **Smart contract developer list.** According to the contract account list, the list of smart contract developers calculates the contribution list of contract developers and the contribution quick rise list to display outstanding contracts developers and Dapps.

6.4 Keyword Query

By providing a keyword or describing the textual information about a smart contract such as its title, author or function, users can find the matching contract from massive smart contracts. Currently, mature and sophisticated algorithms and technologies are available for text searching. By using the natural language processing and inverted index technologies, we can retrieve and sort efficiently in the index database for massive smart contracts. This involves the following key technologies:

1. Topic-oriented distributed crawler technology
2. Multilingual word breaking technology: word breaking is relatively simple for western words. For the word breaking of Chinese words, these algorithms are available: positive maximum matching, negative maximum matching, shortest path word breaking and statistical word breaking.
3. Search term correction and semantic comprehension
4. Inverted index and distributed searching architecture
5. Ranking algorithm to sort the search results

Among these technologies, the ranking algorithm will be designed in combination with Nebulas Rank. Specifically, we use intra-user transfers in the blockchain world as an analogy to webpage reference relations in the Internet world to create the blockchain transaction graph. Then, calculate the NR rank of non-contract accounts by using the NR ranking algorithm described in Section 2.3, calculate the ranks of those contracts by using the contract ranking algorithm described in Section 4.2, and lastly use the calculation result for search result sorting.

6.5 Searching for Similar Smart Contracts

For developers and certain users, they may want to search for smart contracts with similar functions according to the code fragment of a contract. Being different from regular keyword searching, code similarity has its particularity. To implement the searching function for similar smart contracts, we need to use a certain algorithm to measure the code similarity in a number or percentage.

In today's academia, code similarity algorithms are mainly categorized into the string edit distance, token sequence similarity, abstract syntax tree similarity and program dependency graph similarity. These algorithms describe the similarity in terms of code text, structure and syntax from different dimensions. By combining these 4 major algorithms, we put forward 12 features of the code similarity of Nebulas contracts, such as Skeleton Tree, Type Signature and Library Calls. For details, see Appendix B.

Similar to search results of keywords, search results of smart contracts also use the same contract ranking algorithm to sort final results.

7 Fundamental Services and Development Tools

7.1 Domain Name Service

Due to the anonymity of blockchains, account addresses are long and meaningless strings, which are not user-friendly. For this reason, users are prone to misoperations like money loss caused by unintentional funding or the interaction with incorrect objects. In other words, by using domain names that are easy to remember, users can gain better experience. By using smart contracts, the Nebulas development team will implement a DNS-like domain system named Nebulas Name Service (NNS) on the chain while ensuring that it is unrestricted, free and open. Any third-party developers can implement their own domain name resolution services independently or based on NNS.

For example, Alice applied for the domain name "alice.nns" for her account address 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40. To transfer money to Alice, Bob simply needs to enter "alice.nns" in payee information so that the money will be transferred to the correct payee address through the NNS service.

NNS application rules are as follows:

- Top-level sub-domain names will be reserved and unavailable for application, such as *.nns, *.com, *.org and *.edu. Thus, users can only apply for second-level sub-domain names.
- Once the NNS service is active, users can query domain names for availability. For a vacant domain name, users can bid for it through a smart contract. The bidding process is open so that any user can query for others' bids and update their own bids anytime.
- Once the bidding period expires, the highest bidder wins the domain name and the smart contract locks the user's bidding funds. The validity period of the domain name is one year. One year later, the user can freely determine whether to renew or not. If yes, the validity period will be extended for another year. If no, the bidding funds will be refunded to the user's account and the domain name will be released as available again.
- Users can give up the ownership of a domain name at any time. In this case, the bidding funds will be automatically refunded to the user's account and the domain name will be released as available again after domain data is cleared.
- Users can transfer the ownership of a domain name with or without compensation. Nebulas does not intervene in any transactions of domain names.

7.2 Lightning Network

Currently, all public blockchain networks are faced with system scalability challenges. For example, Bitcoin network is only able to process 7 transactions per second and Ethereum is able to process 15 transactions per second. By introducing PoS-like consensus algorithms, mining calculation can be avoided

and the consensus speed can be improved dramatically. However, public blockchains are still greatly challenged by massive micro-payment scenarios in the real world. Put forward in February 2015, the lightning network [47] was designed to set up a channel network for micro-payment between transaction parties, so that large amounts of payments between the parties can be confirmed off the blockchain directly, repeatedly, frequently and bi-directionally in the netting method. When a transaction needs to be settled, the final result will be submitted to the blockchains for confirmation. Theoretically, this can achieve millions of transfers per second. If no point-to-point payment channel is available between the parties, a payment path connecting both parties and consisting of multiple payment channels can also be used to achieve reliable fund transfer between the parties. The lightning network has gone through the PoC phase on both Bitcoin and Ethereum.

Nebulas implements the lightning network as the infrastructure of blockchains and offers flexible design. Any third-party developers can use the basic service of lightning network to develop applications for frequent transaction scenarios on Nebulas. In addition, Nebulas will launch the world's first wallet app that supports the lightning network.

7.3 Developer Tools

A complete set of developer tools are critical to blockchain app developers. So far, developer tool chains for pubic blockchains are incomplete, imposing great challenges to most developers. Nebulas development team will provide a rich set of developer tools, including independent development IDE for smart contracts, block browser, plugins for various popular IDEs (including Eclipse, JetBrains, Visual Studio, Sublime Text, VIM and Atom), debugger, simulator, formal verification tool for smart contracts, background SDKs for various advanced languages, and SDKs for mobile ends.

8 Nebulas Token NAS

The Nebulas network has its own built-in token, NAS. NAS plays two roles in the network. First, as the original money in the network, NAS provides asset liquidity among users, and functions as the incentive token for PoD bookkeepers and DIP. Second, NAS will be charged as the calculation fee for running smart contracts. The minimum unit of NAS is 10^{-18} NAS.

Originally, NAS was sold as ERC20 token on the Ethereum platform, with the maximum NAS aggregate of $X = 10^8$. The issuance modes are as follows:

1. **Community Construction:** Under the direction of the Nebulas sponsor team, $80\%X$ tokens will be used for Nebulas community construction, including ecological incubation and incentive of the Nebulas community blockchain app (DApp), construction of developer community, business and industrial cooperation, marketing and promotion, academic research, educational investment, laws and regulations, and investment in communities and organizations. Specifically, $5\%X$ will be sold to community impact investors, $5\%X$ as Nebulas community development fund and $70\%X$ for reservation.
2. **Sponsor and Development Team Incentives:** Throughout the development of Nebulas, the sponsor and development teams will continuously make human and material resource contributions in aspects of project organizational structure, technology research and development and ecological operation. As for token allocation, $20\%X$ is reserved for team incentive purpose. This part of NAS is initially locked and will be unlocked one year after the completion of the first NAS sale to the community, and gradually distributed to the sponsor and development teams in three years

After the official launch of Nebulas network, any users with Ethereum ERC20 NAS tokens can claim equivalent NAS of Nebulas network using related credential, and the Ethereum ERC20 NAS tokens will be reclaimed. With the evolution of Nebulas network, NAS will grow in the following way:

1. **PoD incentive:** $3\%X$ incentive NAS per year for bookkeepers;
2. **DIP incentive:** $1\%X$ incentive NAS per year for outstanding smart contract developers.

9 Conclusion

What We Hold

From a highly abstract perspective, blockchain is **the right confirmation of data in a decentralized way**, and tokens function as **the carrier of right confirmation value**. The Internet solves the communication of data, while blockchain further solves the right confirmation of data. For the first time ever, blockchain translates public data into private data which will no longer be analyzed and utilized arbitrarily by large enterprises such as Google, Amazon and Facebook.

The essence of blockchain represented by public blockchain is **"Community + Token + Application"**. Community is essentially a from-bottom-to-top ecosystem adhering to the idea of openness, open source, sharing and non-profitability, which is completely different from existing from-top-to-bottom business ecosystems. Token is the carrier of right confirmation value. There will be more scenarios in the future than those used only for attributes of virtual currency and electronic cash. Application simply refer to the technological implementation of blockchain application scenarios. Without the combination with the aforementioned two factors, application alone **cannot fully reflect the charm of blockchain systems**.

The blockchain system represented by public blockchain is the future of blockchain, as its **"trustless"** and **"permissionless"** attributes are the actual value of blockchain systems. On the contrary, most consortium blockchain/enterprise blockchain are **"trust-based"** and **"permission-based"**, which means they cannot break existing patterns and are considered as improved innovations. While public blockchain systems overturn existing cooperation relations and are considered **disruptive innovations**, reflecting the maximum value of blockchain.

What We Are Committed To

Be as the first blockchain search engine around the world, Nebulas is committed to **exploring hidden dimensions of blockchain value** and building value-based blockchain operating systems, search engines and other related extensive apps.

With this commitment, we put forward **Nebulas Rank** to set up the measure of value of the blockchain world, design **Nebulas Force** to empower the self-evolving of blockchains, develop **Developer Incentive Protocol** and **Proof of Devotion** to motivate the upgrade of blockchain value, and construct **Nebulas Search Engine** to help users explore other dimensions of blockchain value.

What We Believe

The ongoing scientific and technological evolving will lead us to a better life with a higher level of **freedom and equality**. As one of the major technologies, blockchain will gradually give full play to its advantages. Being part of this evolving is our greatest happiness and accomplishment.

Similar to the Internet, blockchains will also enter a phase of explosive users/apps. Blockchain

technology will become the **base protocol** of the next-generation "smart network", and the number of users will reach or even go beyond one billion in the next 5 to 10 years. Significant opportunities and challenges will both emerge in the next five years.

Facing the tremendous ecosystem in the future, ask not what blockchain can do for you, ask what you can do for blockchain. Because **blockchain is an organism and economy**. We are glad to share these with all of you in the exploration of blockchain technologies.

References

[1] Luke Anderson *et al.* "New kids on the block: an analysis of modern blockchains." In: (2016). arXiv: 1606.06530. URL: http://arxiv.org/abs/1606.06530.

[2] Annika Baumann, Benjamin Fabian, and Matthias Lischke. "Exploring the Bitcoin network." In: *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies* 1 (2014), pp. 369–374. ISSN: 9789897580239 (ISBN). DOI: 10.5220/0004937303690374. URL: https://www.engineeringvillage.com/blog/document.url?mid=cpx%7B%5C_%7D9ce5505146fd48dcbdM557010178163125%7B%5C&%7Ddatabase=cpx.

[3] Morten L Bech and Enghin Atalay. "The topology of the Federal Funds markets." In: *Economic Policy Review* 14.2 (2008).

[4] Phillip Bonacich. "Factoring and weighting approaches to status scores and clique identification." In: *Journal of Mathematical Sociology* 2.1 (1972), pp. 113–120.

[5] Stephen P. Borgatti. "Centrality and network flow." In: *Social Networks* 27.1 (2005), pp. 55–71. ISSN: 03788733. DOI: 10.1016/j.socnet.2004.11.008.

[6] Stephen P. Borgatti and Martin G. Everett. "A Graph-theoretic perspective on centrality." In: *Social Networks* 28.4 (2006), pp. 466–484. ISSN: 03788733. DOI: 10.1016/j.socnet.2005.11.005.

[7] Michael Boss, Martin Summer, and Stefan Thurner. "Contagion Flow Through Banking Networks." In: *Lecture Notes in Computer Science 3038* (2004), pp. 1070–1077. ISSN: 03029743. DOI: 10.1016/j.jfi.2008.06.003. arXiv: 0403167v1 [arXiv:cond-mat].

[8] Michael Boss *et al.* "The Network Topology of the Interbank Market." In: *Quantitative Finance* 4.6 (2004), pp. 677–684. ISSN: 1469-7688. DOI: 10.1080/14697680400020325. arXiv: 0309582 [cond-mat]. URL: http://arxiv.org/abs/cond-mat/0309582.

[9] Sergey Brin and Lawrence Page. "Reprint of: The anatomy of a large-scale hypertextual web search engine." In: *Computer Networks* 56.18 (2012), pp. 3825–3833. ISSN: 13891286. DOI: 10.1016/j.comnet.2012.10.007. arXiv: 1111.6189v1.

[10] Vitalik Buterin. "Ethereum: A next-generation smart contract and decentralized application platform." In: *URL https://github. com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper* (2014).

[11] Vitalik Buterin *et al. Ethereum white paper*. 2013.

[12] Lijun Chang *et al.* "pSCAN: Fast and Exact Structural Graph Clustering." In: *IEEE Transactions on Knowledge and Data Engineering* 29.2 (2017), pp. 387–401.

[13] Tao Hung Chang and Davor Svetinovic. "Data Analysis of Digital Currency Networks: Namecoin Case Study." In: *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS* (2017), pp. 122–125. DOI: 10.1109/ICECCS.2016.023.

[14] Duan Bing Chen *et al.* "Identifying influential nodes in large-scale directed networks: The role of clustering." In: *PLoS ONE* 8.10 (2013), pp. 1–10. ISSN: 19326203. DOI: 10.1371/journal.pone.0077455.

[15] Michel Chilowicz, Etienne Duris, and Gilles Roussel. "Syntax tree fingerprinting for source code similarity detection." In: *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE. 2009, pp. 243–247.

[16] *Etherscan - The Ethereum Block Explorer*. https://etherscan.io/. Accessed: 2017-08-01.

[17] Giorgio Fagiolo. "The International-Trade Network: Gravity Equations and Topological Properties." In: (2009). arXiv: 0908.2086. URL: http://arxiv.org/abs/0908.2086.

[18] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. "The program dependence graph and its use in optimization." In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 9.3 (1987), pp. 319–349.

[19] Danno Ferrin. "A Preliminary Field Guide for Bitcoin Transaction Patterns." In: *Texas Bitcoin Conference* (2015). URL: http://texasbitcoinconference.com.

[20] Michael Fleder, Michael S. Kester, and Sudeep Pillai. "Bitcoin Transaction Graph Analysis." In: *… -Transaction-Graph-Analysis. Pdf"…* (2015), pp. 1–8. arXiv: 1502.01657. URL: http://arxiv.org/abs/1502.0165%7B%5C%%7D5Cnhttp://people.csail.mit.edu/spillai/data/papers/bitcoin-project-paper.pdf%7B%5C%%7D5Cnhttp://arxiv.org/abs/1502.00165%7B%5C%%7D5Cnhttp://arxiv.org/abs/1502.01657.

[21] L Freeman. "A set of measures of centrality: I. Conceptual clarification." In: *Soc. Networks* 1 (1979), pp. 215–239.

[22] Linton C Freeman. "A set of measures of centrality based on betweenness." In: *Sociometry* (1977), pp. 35–41.

[23] Linton C Freeman. "Centrality in social networks conceptual clarification." In: *Social networks* 1.3 (1978), pp. 215–239.

[24] Linton C Freeman, Stephen P Borgatti, and Douglas R White. "Centrality in valued graphs: A measure of betweenness based on network flow." In: *Social networks* 13.2 (1991), pp. 141–154.

[25] Sudipto Guha *et al.* "Approximate XML joins." In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM. 2002, pp. 287–298.

[26] Bernhard Haslhofer, Roman Karl, and Erwin Filtz. "O Bitcoin Where Art Thou? Insight into Large-Scale Transaction Graphs." In: ().

[27] Leo Katz. "A new status index derived from sociometric analysis." In: *Psychometrika* 18.1 (1953), pp. 39–43.

[28] S King and S Nadal. "Peercoin–Secure & Sustainable Cryptocoin." In: *Aug-2012 [Online]. Available: https://peercoin.net/whitepaper* ().

[29] Jon M Kleinberg. "Authoritative sources in a hyperlinked environment." In: *Journal of the ACM (JACM)* 46.5 (1999), pp. 604–632.

[30] Lothar Krempel. "Exploring the Dynamics of International Trade by Combining the." In: December (2002), pp. 1–22.

[31] Qian Li *et al.* "Identifying influential spreaders by weighted LeaderRank." In: *Physica A: Statistical Mechanics and its Applications* 404 (2014), pp. 47–55. ISSN: 03784371. DOI: 10.1016/j.physa.2014.02.041. arXiv: arXiv:1306.5042v1.

[32] *llvm*. https://llvm.org/. Accessed: 2017-08-01.

[33] Linyuan Lü *et al.* "Vital nodes identification in complex networks." In: *Physics Reports* 650 (2016), pp. 1–63. ISSN: 03701573. DOI: 10.1016/j.physrep.2016.06.007. arXiv: 1607.01134.

[34] Sarah Meiklejohn *et al.* "A fistful of Bitcoins: Characterizing payments among men with no names." In: *Proceedings of the Internet Measurement Conference - IMC '13* 6 (2013), pp. 127–140. ISSN: 15577317. DOI: 10.1145/2504730.2504747. URL: http://dl.acm.org/citation.cfm?id=2504730.2504747.

[35] *Minimal Slashing Conditions*. https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c. Accessed: 2017-08-01.

[36] L Morten, J Robert, and E Walter. "The topology of interbank payment flows." In: (2006).

[37] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." In: *Www.Bitcoin.Org* (2008), p. 9. ISSN: 09254560. DOI: 10.1007/s10838-008-9062-0. arXiv: 43543534534v343453. URL: https://bitcoin.org/bitcoin.pdf.

[38] *NEM Technical Reference*. http://nem.io/NEM_techRef.pdf. Accessed: 2017-08-01.

[39] Mark Newman. *Networks: an introduction*. Oxford university press, 2010.

[40] Mark EJ Newman. "A measure of betweenness centrality based on random walks." In: *Social networks* 27.1 (2005), pp. 39–54.

[41] Dá Niel Kondor *et al.* "Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network." In: *PLoS ONE* 9.2 (2014). DOI: 10.1371/journal.pone.0086197.

[42] Athanasios N. Nikolakopoulos and John D. Garofalakis. "NCDawareRank." In: *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13* February 2013 (2013), p. 143. DOI: 10.1145/2433396.2433415. URL: http://dl.acm.org/citation.cfm?doid=2433396.2433415.

[43] Jae Dong Noh and Heiko Rieger. "Random walks on complex networks." In: *Physical review letters* 92.11 (2004), p. 118701.

[44] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. "Structure and Anonymity of the Bitcoin Transaction Graph." In: *Future Internet* 5.2 (2013), pp. 237–250. ISSN: 1999-5903. DOI: 10.3390/fi5020237. URL: http://www.mdpi.com/1999-5903/5/2/237/.

[45] Lawrence Page *et al. The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[46] Thai Pham and Steven Lee. "Anomaly detection in bitcoin network using unsupervised learning methods." In: *arXiv preprint arXiv:1611.03941* (2016).

[47] Joseph Poon and Thaddeus Dryja. "The bitcoin lightning network: Scalable off-chain instant payments." In: *Technical Report (draft)* (2015).

[48] Marc Pröpper, Iman van Lelyveld, and Ronald Heijmans. "Towards a network description of interbank payment flows." In: (2008).

[49] Dorit Ron and Adi Shamir. "Quantitative Analysis of the Full Bitcoin Transaction Graph." In: ().

[50] Gert Sabidussi. "The centrality index of a graph." In: *Psychometrika* 31.4 (1966), pp. 581–603.

[51] Computer Science and The Technion. "The Stochastic Approach for Link-Structure Analysis (SALSA) and the TKC E ect." In: (2001).

[52] M. Ángeles Serrano, Marián Boguñá, and Alessandro Vespignani. "Patterns of dominant flows in the world trade web." In: *Journal of Economic Interaction and Coordination* 2.2 (2007), pp. 111–124. ISSN: 1860711X. DOI: 10.1007/s11403-007-0026-y. arXiv: 0704.1225.

[53] Ma Angeles Serrano and Marián Boguñá. "Topology of the world trade web." In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 68.1 Pt 2 (2003), p. 015101. ISSN: 1063-651X. DOI: 10.1103/PhysRevE.68.015101. arXiv: 0301015 [cond-mat].

[54] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. "SCAN++: efficient algorithm for finding clusters, hubs and outliers on large-scale graphs." In: *Proceedings of the VLDB Endowment* 8.11 (2015), pp. 1178–1189.

[55] *The Stage 1 Casper Contract*. https://github.com/ethereum/casper/. Accessed: 2017-08-01.

[56] Florian Tschorsch and Björn Scheuermann. "Bitcoin and Beyond : A Technical Survey on Decentralized Digital Currencies." In: *IEEE COMMUNICATIONS SURVEYS & TUTORIALS* PP.99 (2015), pp. 1–1. ISSN: 1553-877X. DOI: doi:10.1109/COMST.2016.2535718.

[57] Xiaowei Xu *et al.* "Scan: a structural clustering algorithm for networks." In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2007, pp. 824–833.

[58] Kaizhong Zhang and Dennis Shasha. "Simple fast algorithms for the editing distance between trees and related problems." In: *SIAM journal on computing* 18.6 (1989), pp. 1245–1262.

Appendix A Nebulas Account Address Design

A.1 Address Checksum

Similar to Bitcoin and Ethereum, Nebulas also adopts elliptic curve algorithm as its basic encryption algorithm for Nebulas accounts. A user's private key is a randomly generated 256-bit binary number, based on which a 64-byte public key can be generated via elliptic curve multiplication. Bitcoin and Ethereum addresses are computed by public key via the deterministic Hash algorithm, and the difference between them lies in: Bitcoin address has the checksum design aiming to prevent a user from sending Bitcoins to a wrong user account accidentally due to entry of several incorrect characters; while Ethereum doesn't have such checksum design.

We believe that checksum design is reasonable from the perspective of users, so Nebulas address also includes checksum, for which the specific calculation method is provided as follows:

```
Data = sha3_256(Public Key)[-20:]
CheckSum = sha3_256(Data)[0:4]
Address = "0x" + Hex(Data + CheckSum)
```

The last 20 bytes of SHA3-256 digest of a public key serve as the major component of an address, for which another SHA3-256 digest should be conducted and the first 4 bytes should be used as a checksum, which is equivalent to the practice of adding a 4-byte checksum to the end of an Ethereum address. For example:

1. The standard address of Alice's Ethereum wallet is 0xdf4d22611412132d3e9bd322f82e2940674ec1bc;
2. The final address of Nebulas Wallet should be:
   0xdf4d22611412132d3e9bd322f82e2940674ec1bc**03b20e40**

An Ethereum address contains a total of 42 characters including the prefix "0x", while a Nebulas address contains a total of 50 characters after being added with a checksum composed of 8 characters.

A.2 Extended Address Verification

In addition to standard address with 50 characters, we also support extended address in order to ensure the security of transfers conducted by users. The traditional bank transfer design is used for reference: In the process of a bank transfer, bank card number of the remittee should be verified, in addition to which the remitter must enter the name of the remittee. The transfer can be correctly processed only when the bank card number and the name match each other. The generating algorithm for extended address is described as follows:

```
Data = sha3_256(Public Key)[-20:]
CheckSum = sha3_256(Data)[0:4]
```

```
Address = "0x" + Hex(Data + CheckSum)


ExtData = Utf8Bytes({Nickname or any string})
ExtHash = sha3_256(Data + ExtData)[0:2]
ExtAddress = Address + Hex(ExtHash)
```

An extended address is generated through addition of 2-byte extended verification to the end of a standard address and contains a total of 54 characters. Addition of extended information allows the addition of another element verification to the Nebulas Wallet APP. For example:

1. The standard address of Alice's wallet is 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40, and the extended address after addition of the nickname "alice" should be 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40**e345**.
2. Alice tells Bob the extended address 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40e345 and her nickname **alice**.
3. Bob enters 0xdf4d22611412132d3e9bd322f82e2940674ec1bc03b20e40e345 and alice in the Wallet App.
4. The Wallet App verifies the consistency between the wallet address and the nickname in order to avoid the circumstance that Bob enters the account number of another user by mistake.

Both of standard address and extended address can be used for transfer. However, our Nebulas Wallet APP will require extended address mandatorily. Therefore, a user is required to provide nickname of the remittee, based on which the consistency between the nickname and address will be verified in order to further enhance security check.

Appendix B Search for Similar Smart Contracts

The difficulty in code similarity lies in structural features of high-level language and diversity in the forms of logical expression of smart contracts. At present, there are various schools of code similarity algorithm in the academic circles, which are generally described as follows:

- **Edit Distance between Character Strings**

  Both of the entered query code and candidate source code are deemed as texts. Edit distance between two character strings is used for measuring similarity between them. Edit Distance refers to the minimum number of editing operations required for converting one character string into the other character string. Permitted editing operations include replacement of one character with another character, i.e. insertion of a character and deletion of a character. Generally speaking, the shorter the edit distance, the higher the similarity between two character strings. This algorithm based on edit distance between character strings can be used not only for source code comparison but also in intermediate representation or even machine language. For the purpose of improving the robustness of the algorithm based on edit distance between character strings, a certain degree of conversion of the source code without any semantic change will be conducted, such as removal of blank character, removal of annotation, replacement of names of all local variables with '$', normalized expression of algebraic expression, etc. This algorithm is characterized by fast speed, conciseness and high efficiency. However, its adaptability to complex programs is relatively poor and doesn't take syntax and organizational structure of code into consideration.

- **Token Sequence**

  Token sequence representation method refers to the conversion of the entered source code into a Token sequence through the analysis by a lexical analyzer. Similarity between two programs is similarity between two Token sequences, so the longest common substring or correlation matching algorithm (suffix tree matching algorithm) may be used to measure the degree of similarity between two programs, through which code segments with different syntaxes but similar functions can be detected. However, this method conceals organizational structure of programs when measuring the similarity between two programs.

- **Abstract Syntax Tree (AST)**

  AST is an intermediate expression form after syntactic analysis on a source code is conducted, based on which the similarity between two programs can be measured through comparison between one subtree and another subtree. For measurement of the similarity between two trees, the tree edit distance algorithm [58] may be used. The accurate tree edit distance algorithm is relatively complex and Literature [25] provides an approximate fast algorithm. According to Literature [15], syntax tree should be subject to Hash fingerprint in order to enable the syntax tree comparison algorithm to conduct high-efficiency searching on massive datasets.

- **Program Dependency Graph (PDG)**

PDG [18] can represent internal data and control dependency relationship of a program and analyze program code at the semantic level. Similar code protocol becomes search of isomorphic subgraphs, which is the NP-complete problem and requires a very complex algorithm, so only some approximate algorithms are available currently.

We believe that the abovementioned algorithms describe similarity between codes in text, structure and syntax at different dimensions. Source Forager [27] provides a great engineering thought: Indexes of similarity at various dimensions are depicted as different features, each of which represents code similarity measurement from a specific aspect. Finally, vector similarity is used to conduct overall similarity measurement. This method integrates the advantages of the abovementioned algorithms. This thought is also used by Nebulas for reference in realizing search of similarity among smart contracts. We deem function as the fundamental unit of code search among smart contracts.

Table 3 defines the candidate code similarity features. Next, we are going to describe definition of each feature and the function for calculating their similarity:

Table 3: Code Similarity Feature Family Table

Feature-Class	Brief Description
Type–Operation Coupling	types used and operations performed on the types
Skeleton Tree	structure of loops and conditionals
Decorated Skeleton Tree	structure of loops, conditionals, and operations
3 Graph CFG BFS	CFG subgraphs of size 3, BFS used for generating subgraphs
4 Graph CFG BFS	CFG subgraphs of size 4, BFS used for generating subgraphs
3 Graph CFG DFS	CFG subgraphs of size 3, DFS used for generating subgraphs
4 Graph CFG DFS	CFG subgraphs of size 4, DFS used for generating subgraphs
Library Calls	calls made to libraries
Type Signature	input types and the return type
Local Types	types of local variables
Numeric Literals	numeric data constants used
String Literals	string data constants used

- **Type–Operation Coupling**

  This feature is a two-tuple set. Two tuples contain type of variable and operator of type of variable, namely the (type, operation) pair. Generally, primitive data type should be paired with arithmetic operator, logical operator and relational operator, such as $(int, \geq)$; custom data type (such as struct) should be paired with member function, such as (Bar, .foo), indicating that field "foo" of data type "Bar" is accessed. Based on this method, all operations on variables in the code body of

a function can be changed into two-tuples. After repetition removal, A two-tuple sequence is used to reflect the Type–Operation Coupling feature of this code segment. We believe that codes with similar functions should have similar variable operation sets. However, we are not concerned with the order of the two tuples, so this feature loses the logical structure information of code and thus can only represent feature of code partially.

Similarity among Type–Operation Coupling features can be defined by Jacobian similarity, namely that if two sets S_1 and S_2 are given, Jacobian similarity can be defined with the following formula:

$$sim_{Jacc(S_1,S_2)} = \frac{\mid S_1 \bigcap S_2 \mid}{\mid S_1 \bigcup S_2 \mid} \tag{12}$$

- **Skeleton Tree**

  Code-based abstract syntax tree. However, only loop (for, while, do...while) and conditional statement (if...else) are reserved, and all the other nodes are removed from the tree. We believe that codes with similar functions should be similar in structure of loop and conditional statement.

  Similarity calculation for skeleton tree is based on edit distance between two trees. d_r is defined as the estimated edit distance between two trees and is only determined by size of tree, namely that:

$$d_r(T_1, T2) = \frac{\mid size(T_1) - size(T_2) \mid}{max(size(T_1), size(T_2))} \tag{13}$$

  D_T is assumed as the threshold value of edit distance and set as 0.5. We can further acquire the formula for calculation of approximate edit distance between two trees:

$$d_t(T_1, T2) = \begin{cases} d_r(T_1, T2) & if\ d_r(T_1, T2) \geq D_T \\ \dfrac{max\begin{pmatrix} ed(pre(T_1),\ pre(T_2)), \\ ed(post(T_1),\ post(T_2)) \end{pmatrix}}{max(size(T_1), size(T_2))} & otherwise \end{cases} \tag{14}$$

  pre(T) represents preorder traversal sequence of tree; post(T) represents postorder traversal sequence of tree; $ed(S_1, S_2)$ represents edit distance between S_1 and S_2. Similarity between two skeleton trees can be calculated with the following formula:

$$sim_{Tree}(T_1, T2) = 1 - d_t(T_1, T2) \tag{15}$$

- **Decorated Skeleton Tree**

  Decorated Skeleton Tree is similar to Skeleton Tree. In addition to loop and branch node, most operators (such as +, -, <) are reserved. However, assignment operators are removed because most of these operators are noises.

- **K-Subgraphs of CFG**

  Realized based on k-subgraph of CFG of a function. k-subgraph should be defined with the following

method: A CFG and a specific node should be given, based on which we should conduct breadth-first search (BFS) or depth-first search (DFS) until number of traversed nodes reaches k, when the formed subgraph should be k-subgraph. If number of nodes fails to reach k after finish of traversal, such subgraph should be discarded. Through traversal of each node of CFG, we can acquire all k-subgraphs. For each k-subgraph, k^2 bit integer is used to express it. Refer to Figure 20. All k-subgraphs form one integer set.

3 Graph CFG BFS: k = 3, BFS Traversal

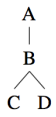
4 Graph CFG BFS: k = 4, BFS Traversal

3 Graph CFG DFS: k = 3, DFS Traversal

4 Graph CFG DFS: k = 4, DFS Traversal

Similarity can be calculated with generalized Jacobian similarity formula: Vectors $\vec{x} = (x_1, x_2, ...x_n)$ and $\vec{y} = (y_1, y_2, ...y_n)$ are given, based on which generalized Jacobian similarity can be defined as:

$$J(\vec{x}, \vec{y}) = \frac{\sum_i min(x_i, y_i)}{\sum_i max(x_i, y_i)} \tag{16}$$



	A	B	C	D
A	0	1	0	0
B	0	0	1	1
C	0	0	0	0
D	0	0	0	0

Fig. 20: 4-graph example: Element of adjacency matrix is a binary string "0100 0011 0000 0000", for which the decimal number is 17152

- **Library Calls**

  If call of any contract from any other library occurs in the contract, addresses of all library contracts will be recorded. Similarity among them will be calculated with Jacobian similarity formula.

- **Type Signature**

  This feature is composed of input parameter type and return parameter type, and similarity between them can be calculated with Jacobian similarity formula. For example, for the following smart contract code, feature of Type Signature of function "getBalance" is vector (address, uint256).

```
contract addressTest {
  function getBalance(address addr) returns (uint) {
        return addr.balance;
  }
}
```

- **Local Types:** This feature is the set of all types of local variables of the function body, for which similarity should be calculated with Jacobian similarity formula.
- **Numeric Literals:** The set of all numeric constants serves as the feature of Numeric Literals, for which similarity should be calculated with Jacobian similarity formula.
- **String Literals:** The set of all string constants serves as the feature of String Literals, for which similarity should be calculated with Jacobian similarity formula.

Feature family can be expanded, so it is convenient to add new features to it. Based on the circumstance that there is a similarity calculation for each feature, we calculate the weighted sum of all features and thus can acquire the final code similarity:

$$sim_{combined}(\vec{A}, \vec{B}) = \frac{\sum_{c=1}^{n_{cl}} sim_c(\vec{A_c},\ \vec{B_c}) \cdot w_c}{\sum_{c=1}^{n_{cl}} w_c} \tag{17}$$

Therein, $\vec{A}$ and $\vec{B}$ are eigenvectors; n_{cl} is number of features in the feature family; sim_c is similarity calculation function specific to feature c; $\vec{A_c}$ and $\vec{B_c}$ are eigenvectors of feature c; w_c is weight of c. Weight can be acquired through machine learning algorithm training based on a large number of test sets.