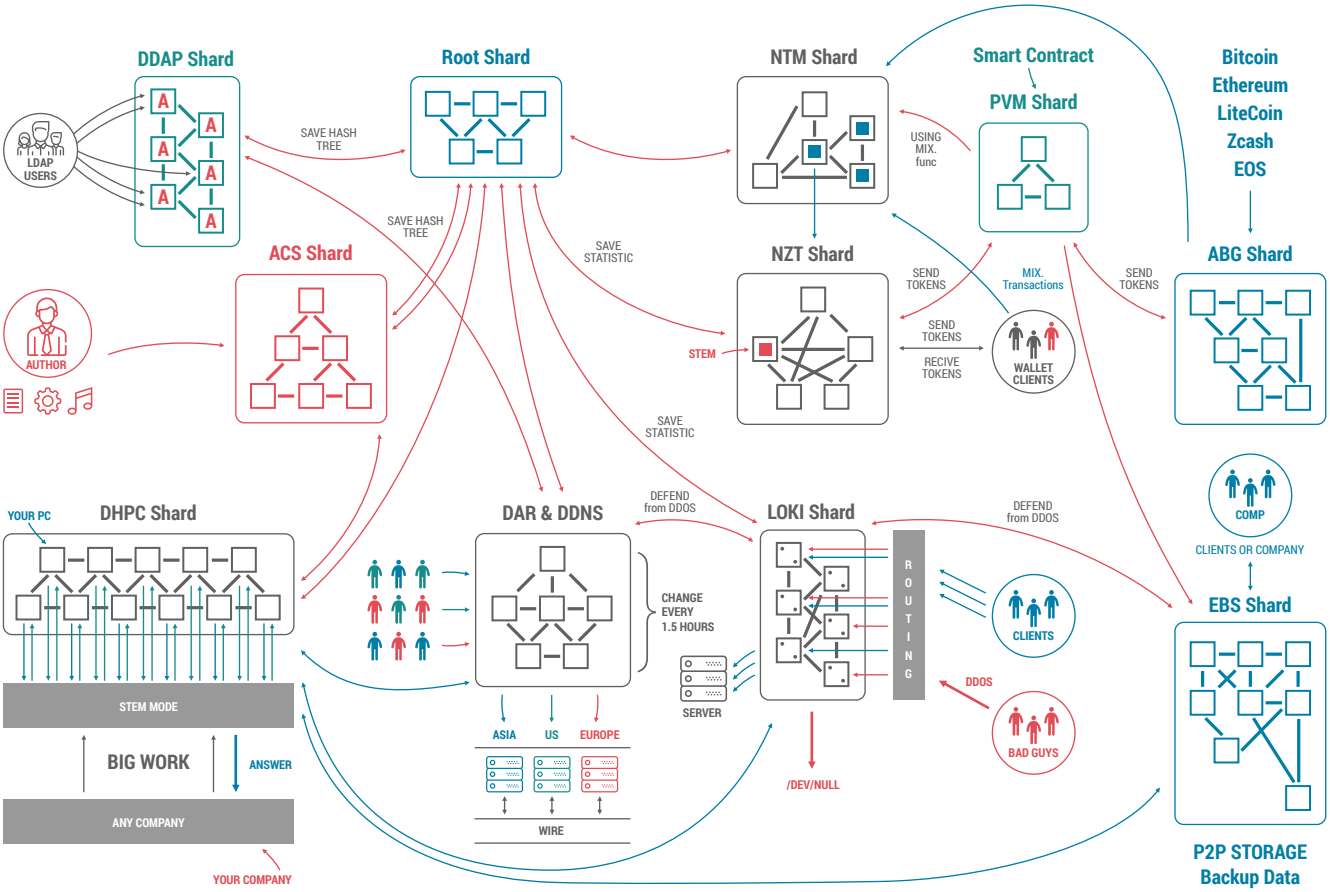

Noosphere
TECHNICAL
WHITEPAPER

Contents:

1 Abstract

NOOSPHERE ECOSYSTEM



Noosphere is a cloud, decentralized blockchain platform that offers a pioneering architecture for building distributed internet services and computing systems using Service Oriented Sharding technology. Each Noosphere shard is a dedicated blockchain service that any user can implement and use. All shards together form a computing ecosystem with the following features: scalability, high speed, decentralization, interoperability and fault tolerance. Noosphere is set to be the first blockchain platform to enable creation of new internet services and computing systems of any complexity as well as to take control of the existing ones.

PLEASE NOTE: CRYPTOGRAPHIC TOKENS REFERRED TO IN THIS WHITE PAPER REFER TO CRYPTOGRAPHIC TOKENS ON A LAUNCHED BLOCKCHAIN THAT ADOPTS THE NOOSPHERE SOFTWARE. THEY DO NOT REFER TO THE ERC-20 COMPATIBLE TOKENS BEING DISTRIBUTED ON THE ETHEREUM BLOCKCHAIN IN CONNECTION WITH THE NOOSPHERE TOKEN DISTRIBUTION.

2 Disclaimer

This Noosphere Technical White Paper v4 is for information purposes only. Noosphere Foundation does not guarantee the accuracy of or the conclusions reached in this white paper, and this white paper is provided “as is”. Noosphere Foundation does not make and expressly disclaims all representations and warranties, express, implied, statutory or otherwise, whatsoever, including, but not limited to: (i) warranties of merchantability, fitness for a particular purpose, suitability, usage, title or non-infringement; (ii) that the contents of this white paper are free from error; and (iii) that such contents will not infringe third-party rights. Noosphere Foundation and its affiliates shall have no liability for damages of any kind arising out of the use, reference to, or reliance on this white paper or any of the content contained herein, even if advised of the possibility of such damages. In no event will Noosphere Foundation or its affiliates be liable to any person or entity for any damages, losses, liabilities, costs or expenses of any kind, whether direct or indirect, consequential, compensatory, incidental, actual, exemplary, punitive or special for the use of, reference to, or reliance on this white paper or any of the content contained herein, including, without limitation, any loss of business, revenues, profits, data, use, goodwill or other intangible losses.

3 Actual Requirements for Blockchain Platforms

Today blockchain technology develops like no other sphere of information technology. The advent of this technology has brought along new venues for developing and improving the existing approaches to data storage and transmission. Nevertheless, development is accompanied with new requirements being put forward by end users: businesses, governments and individuals all over the world. Platforms that fail to meet those have no future.

3.1 Service-oriented architecture

Blockchain possesses a number of advantages appealing to businesses, specifically: tremendous computing power, security, reliability, fault tolerance, simplicity and transparency, however, none of the existing blockchain platforms offers mechanisms to implement private business logic on top of its architecture. Their functionality is limited to a payment function and smart contracts, the two features being narrowly applicable. This significantly bogs down the industry development: we cannot use the existing blockchain platforms to build new services on top of them and transfer the existing ones into them. The need has become evident now for a new service-oriented architecture. Future systems must provide APIs for creating services running on top of a single computer network and enable inter-service data exchange.

3.2 Effective scalability

User traffic and data flow on blockchain networks grow with every passing day. The throughput capacity of popular platforms cannot catch up with the ever growing requirements. Every day sees new ideas and their adaptation algorithms: an increase in the block size, variations of consensus algorithms, the use of side-chain solutions and others. All this will only have a temporary effect, for understandable reasons. New platforms must be designed allowing for the fact that the number of users and data flow will be dozens of times bigger in the future than today. The only way out is efficient scaling. Unlike centralized systems, blockchain makes it easy owing to its decentralization. Division of the input flow and parallel data processing at numerous shards make the system capable of operating with equal efficiency for tens of thousands and tens of millions of users.

3.3 Business-oriented performance

The total hash rate of the bitcoin platform has reached 4 Exa hashes per second. Few supercomputers can boast such performance. However, all these computing resources are expended only to maintain the operation of the obsolete consensus algorithm, PoW. Statistics show that over 90% of companies prefer to use virtual and cloud technology instead of buying their own resources. The use of the resources of blockchain networks for useful computation, in particular for implementing a service-oriented architecture, can attract many more interested users and expand tremendously blockchain application fields.

3.4 Safety and liveness

Decentralized systems have special requirements for their security, ability to recover, and upgrading processes. Platforms without such mechanisms can only change through forking, only to give rise to even more problems for users. It is also worthwhile noting that forks are simply unacceptable in business applications. A special approach must be developed that will enable the system to evolve not only without forks, but also without critical delays. In the absence of such mechanisms, the system will not be competitive and user-friendly.

4 Service-oriented sharding

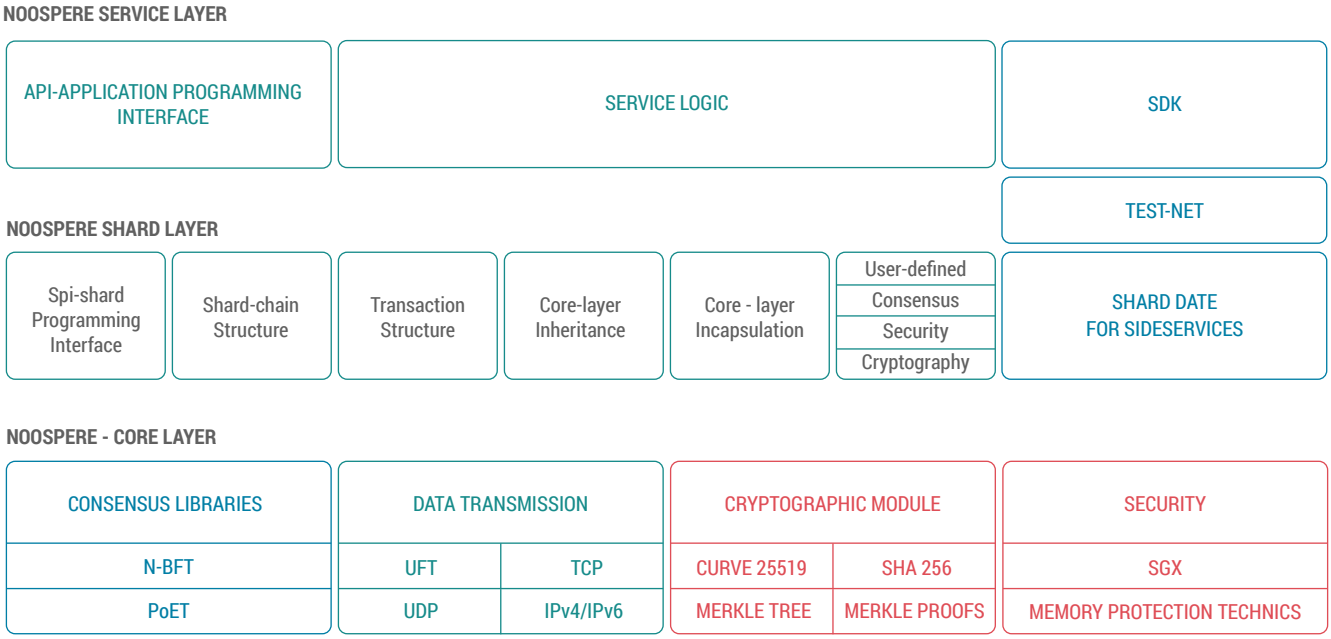
The idea of service-oriented sharding is first mentioned in a research paper titled Service-Oriented Sharding for Blockchains by Adam Efe Gencer. The key difference of this idea is the use of shards in the form of services. Every shard is a special isolated service performing regulated functions, capable of information exchange with other shards (or, in other words, services) within the system. Noosphere is principally different from other sharding-based systems in that it not only supports the standard transaction transfer function and execution of smart contracts but, first and foremost, it supports any service implemented using blockchain technology or using it as a control module. In other words, Noosphere is the pioneering approach to creating the architecture of decentralized internet services and not an alternative replica of the standard sharding ideas. Such services include DNS, DAR, LDAP, Artificial neural networks, routing, proxy, distributed computing, load-distribution sharding, certifying centers, payment functions proper, and business-oriented services with free-format transactions and a block structure for implementing business logic especially needed by companies and corporations unable to maintain computing resources of their own.

Shards for each of such services can be added to the system dynamically and expand the total system functionality. Furthermore, service-oriented sharding makes it easy to implement any side-chains and use any existing third-party blockchain as a shard. In this case, a special service shard has only to perform the function of data routing and maintain data integrity.

There are two types of shards in Noosphere: static and dynamic. A static shard is a system-defined shard that serves to maintain its operability and expand the functions of the core. A dynamic shard is a user-defined shard that can be created by any user for implementing private logic. Any dynamic shard can become static if its functions are critical to the system and it receives the community's approval. One should not think of every individual shard as a separate set of physical machines that maintain its operation. Numerous shards may have a purely virtual structure because they are meant to be managing or controlling entities for shards working under a higher load. While static shards are thought to ensure vertical scalability, dynamic ones do horizontal.

The Noosphere core has no built-in mechanisms for processing smart contracts, anonymizing transactions, balancing load, and others because all these are services and they must be separated as individual shards. This way, Noosphere comes to gain advantages that other platforms lack. The system architecture is flexible and can at any moment change its operation principles by modifying and creating new shards. Furthermore, Noosphere impose no restrictions: anyone can use their own types of smart contracts and their own types of virtual machines to process those contracts, create their own transaction mixers and any other services expanding the functionality of the entire system – and all these are just individual shards.

5 Noosphere anatomy



5.1 Design Principles

5.1.1 KISS – Keep it simple, stupid

The system performs better if it remains simple and clear. This not only lowers the entry barrier, but also enables easy modifications and improvements. The system must be transparent and easy to learn, not complex and formidable, scaring away users and developers.

5.1.2 Be a part of community

The community of developers of open source systems provides phenomenal opportunities for active, by-the-hour system development. People united by the same idea and goals are the strongest driver of progress. Support from these people, including financial, their motivation makes it possible to create truly fundamental platforms capable of competing with closed and commercial organizations.

5.1.3 Flexible is better than solid

One way to build a competitive system that will be able to efficiently develop every day is to keep to a minimum the number of functions built into the core and provide mechanisms for implementing new functionality by means of flexible modules that can easily be loaded into and unloaded from the system.

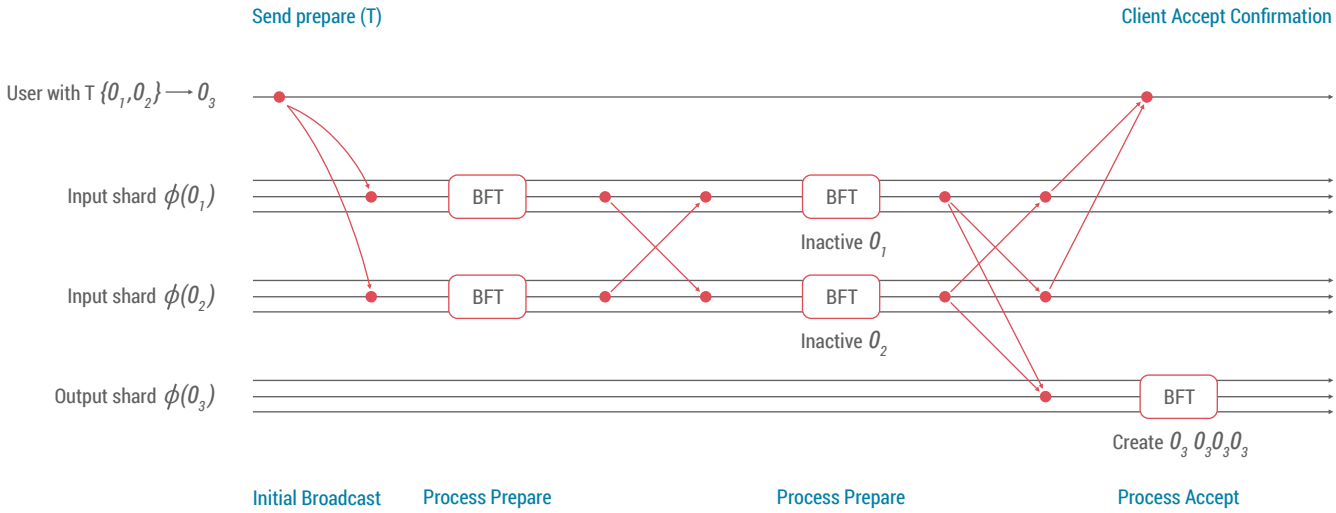
5.1.4 Know your users' requirements

The only right direction for developing a system is a dialogue with users. They alone know what functions are necessary today and what will be necessary tomorrow. In the absence of this communication, a system risks going the wrong way and finding itself redundant and useless.

5.1.5 Safety

One of the first requirements important to a user is absolute security of personal data. Although this lies at the heart of the blockchain philosophy, not many platforms under development pay due attention to it. They go the wrong way simply copying their predecessors. Only with an experienced team of developers, engineers, mathematicians, and cryptographers on board can we develop a system that will fully meet this requirement.

5.2 Sharding



Sharding is at the heart of the Noosphere architecture. The standard method of using shards is separate processing of transactions that are divided according to a certain criterion between servicing shards. This enables a significant increase in the throughput and operating speed of the system compared with systems in which nodes share a single copy of a blockchain. Every shard in the Noosphere platform is a mini blockchain that can work absolutely independently: constant connection with other shards is unnecessary for its operation. At the same time, the main blockchain of the system is also a shard. Thus, the elementary unit of Noosphere is the shard. The structure of each shard is the same, only the functions they perform are different.

5.2.1 Sharding technical

Sharding is a method of horizontal data division. The sharding process itself is divided into three elements:

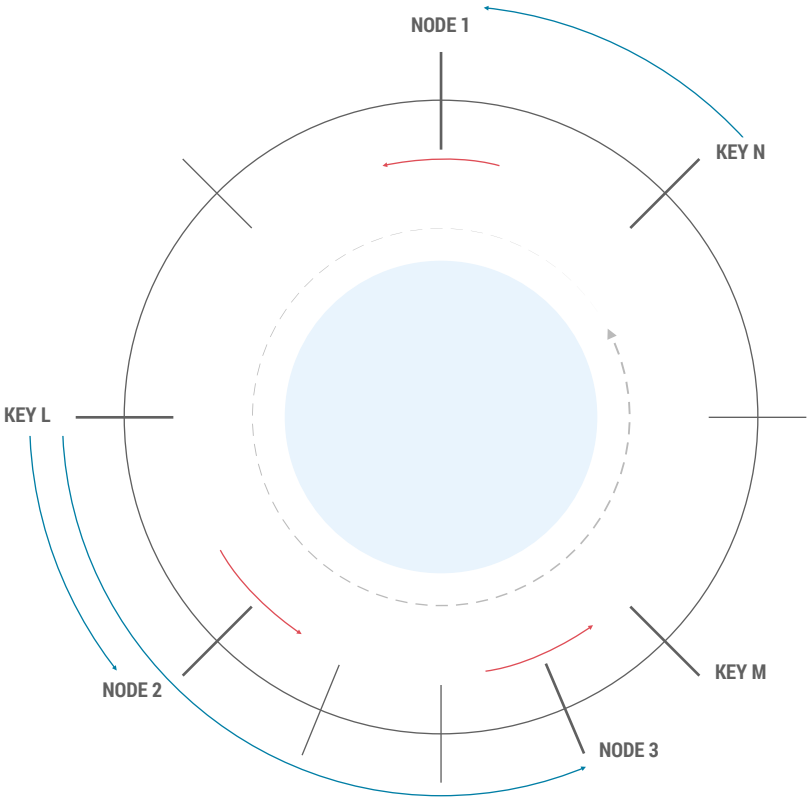
- Selection of the sharding function;
- Re-sharding, or the process of data redistribution;
- Data routing (identification of the physical location of data).

In its simplest form, the sharding function can be presented as $f(x_1, x_2, x_3,..., x_n) = y$, where x_n is a sharding key and y is the shard. The correct choice of sharding keys directly influences the sharding efficiency. Usually these keys consist of the number of nodes available for operation and the identifier of the user owning the data. The sharding function f itself is a special hashing function. This can be a consistent hashing function or an HRW (Highest Random Weight) hashing function. The efficiency of the selected function is manifested at the time of data resharding when n of K nodes are out of order and data must be redistributed between the nodes still online or on the contrary, when network load dramatically increases, so new shards have to be formed for load distribution. Existing approaches to sharding in blockchain platforms omit the importance of this problem and deal only with selection of the hashing function in the belief that network load is constant and the originally selected configuration of shards will not change in the future.

Consistent hashing is a special kind of hashing such that when a hash table is rebuilt, only $\frac{K}{n}$ keys need to be remapped on average, where K is the number of keys, and n is the number of slots. In contrast, in most traditional hash tables, a change in the number of slots causes nearly all keys to be remapped. The use of consistent hashing rids us of the key rehashing operation when the set of active nodes is modified (through addition or removal).

Instead, the entire set of hashed keys of nodes and data is located in a closed ring, so the addition of new nodes enables dynamic load distribution without the need for rehashing. HRW hashing is a similar function, but its potential is much greater, so consistent hashing can be considered a special case in certain situations. HRW hashing also distributes sets of keys in a ring, using a uniform hash function. Unlike consistent hashing, HRW requires no precomputing or storage of keys. An object O_i is assigned to one of n nodes $N_1,...,N_n$ by computing the n hash values $h(O_i, N_j)$ and choosing the node N_k, that yields the highest has function value. If a new node N_{n+1} is added, new object placements or requests will compute $n+1$ hash values and pick the highest of them. If an object already in the system at N_k maps to this new node N_{n+1}, it will be fetched afresh and cached at N_{n+1}. All clients will henceforth obtain it from this node, and the old cached copy at N_k will eventually be replaced by the local cache management algorithm. If N_k is taken offline, its objects will be remapped uniformly to the remaining $n-1$ sites. Variants of the HRW algorithm, such as the use of a skeleton can reduce the time $O(n)$ for object location to $O(\log(n))$ at the cost of less global uniformity of placement. When n is not too large, however, the $O(n)$ placement cost of basic HRW is not likely to be a problem. HRW completely avoids all the overhead and complexity associated with correctly handling multiple keys for each node and associated metadata.

NOOSPHERE
HRW HASHING

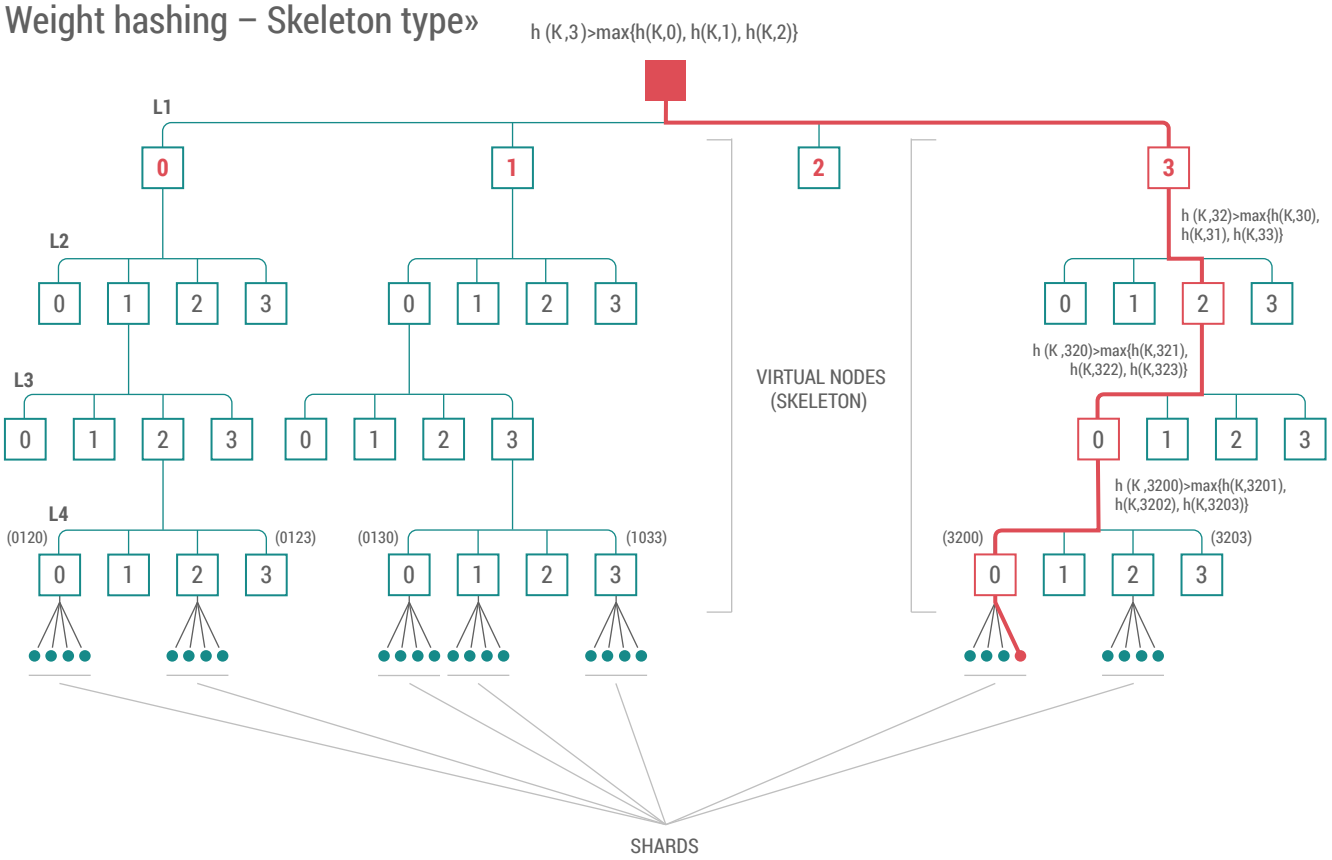


When n is extremely large, an HRW algorithm variant such as the use of a skeleton can significantly reduce object location time. This approach creates a virtual hierarchical structure (called a skeleton), and achieves $O(\log n)$ running time by applying HRW at each level while descending the hierarchy. At the first stage, a constant m is chosen and n nodes are formed in $c=n/m$ shards $S_1 = \{N_1, N_2,..., N_m\}$, $S_2 = \{N_{m+1}, N_{m+2},..., N_{2m}\},...$ At the second stage, a virtual hierarchy is created from these shards placed at the leaves of a tree T of virtual nodes, each with fanout f.

Assuming that the shard size is $m = 4$, the skeleton fanout is $f = 4$, and the total number of real nodes is 128.

Instead of applying HRW to all 128 real nodes, we can first apply HRW to the 32 lowest-tier virtual nodes, selecting one. We then apply HRW to the four real nodes in its shard, and choose the winning node. We only need 32 + 4 = 36 hashes, rather than 128. We can start at any level in the virtual hierarchy,
not just at the root. Starting lower in the hierarchy requires more hashes, but may improve load distribution in the case of failures. Also, the virtual hierarchy need not be stored, but can be created on demand, since the virtual nodes names are simply prefixes of ***base-f representations***. Clearly, T has height $h = O(\log c) = O(\log n)$ since m and f are both constants. The work done at each level is $O(1)$, since f is a constant.
For any given object O, it is clear that the method chooses each shard, and hence each of the n nodes, with equal probability. If the node finally selected is unavailable, we can select a different node within the same shard, in the usual manner. Alternatively, we could go up one or more tiers in the skeleton and select an alternative from among the sibling virtual nodes at that tier, and once again descend the hierarchy to the real nodes, as indicated above. The value of m can be chosen based upon such factors as the anticipated failure rate and the degree of load balancing desired. A higher value of m leads to less load in the event of failure, at the cost of somewhat higher search overhead.

NOOSPHERE «Highest Random Weight hashing – Skeleton type»



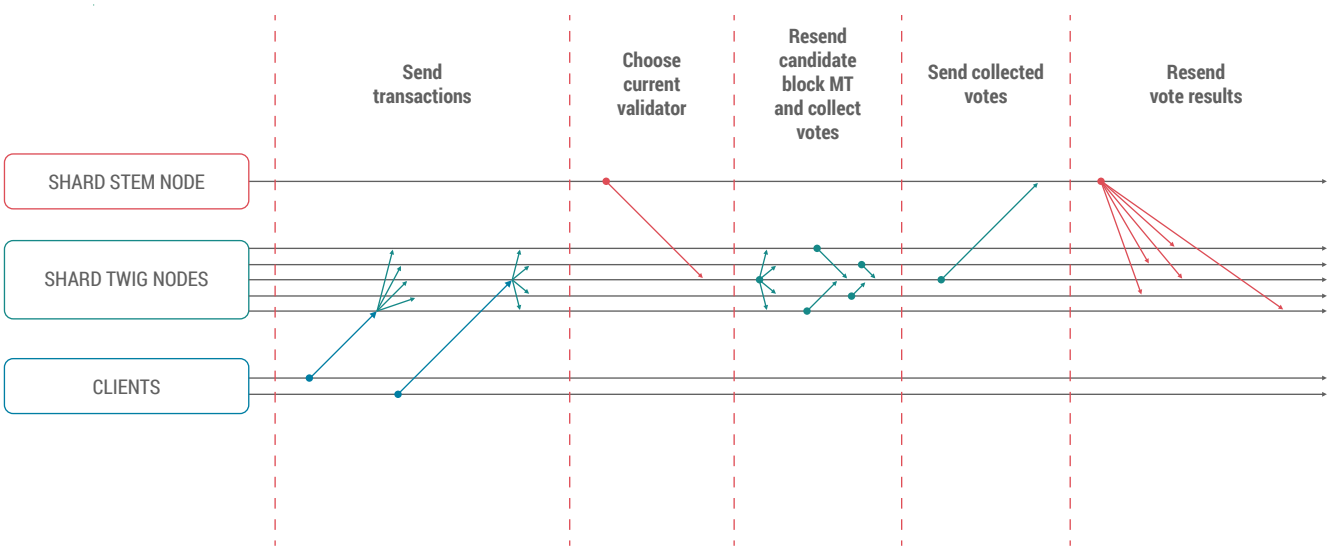
The matter of routing is no less important. It determines the speed of data delivery to the end user and of data exchange between shards. The simplest and least efficient method is to store routing tables at clients. However, this method is deprived of scalability. The options of using separately dedicated coordinators and proxies are impossible on account of the decentralized architecture of blockchain systems. Thus the matter of routing is placed on the nodes of the system themselves. The simplest decision making it possible to find the source of necessary data over the logarithmic time log n is to have nodes store information about their neighbors and about a limited set of nonadjacent nodes. Thus on any request, if the requested data is not stored at the current node, it routes the request to its neighbors (if they have the key) or next to a random nonadjacent node.

5.3 Shard structure

NOOSPHERE ELEMENTS STRUCTURE

TYPE	FUNCTIONS	NEEDED POWER
END CLIENT	• TRANSATIONS • CONTRACTS • VOTING	LOW
TWIG NODE	• SAVE CHAIN • COLLECT TRANSACTIONS • CHECK TRANSACTIONS • BUILD BLOCKS • RESEND TR. & BLOCKS • GATE FUNCTION • VOTING	MEDIUM / HIGH
STEM NODE	• SAVE CHAIN • SYNCHRONIZE • EXCLUDE • CHOOSE	MEDIUM / HIGH
ROOT SHARD	• SAVE ALL CHAINS • GIVE DATA TO WORLD • ROUTING	HIGH

NOOSPHERE SHARD STRUCTURE



Every Noosphere shard consists of a set of nodes with various functions. A node that participates in the consensus algorithm, gathering of transactions, and forming and validation of blocks is a Twig Node. A node that participates in the CBFT consensus algorithm speeding up its operation is a Stem Node. The function of a Stem Node is exclusively auxiliary: the system can function without it, but a different consensus algorithm is used in this case (see the Consensus section for more information). Anyone can become a Twig Node for a shard by filing an application and passing selection (see the Consensus section for more information). A Stem Node is serviced by the creator of the shard if it is necessary, for its operation, to use the quick and efficient CBFT consensus algorithm. If the data processing speed requirements are not as important, a shard can function independently, using PoET without a Stem Node.

5.4 Root shard

The main shard of Noosphere is called the Root Shard. It coordinates the operation of other shards when they exchange data. The co-ordination is based on the following functions:

1. Confirmation of the validity of data received from another shard;
2. Provision of a list of active shards;
3. Provision of a list of active Twig Nodes of a shard;
4. Provision of service data about the shard.

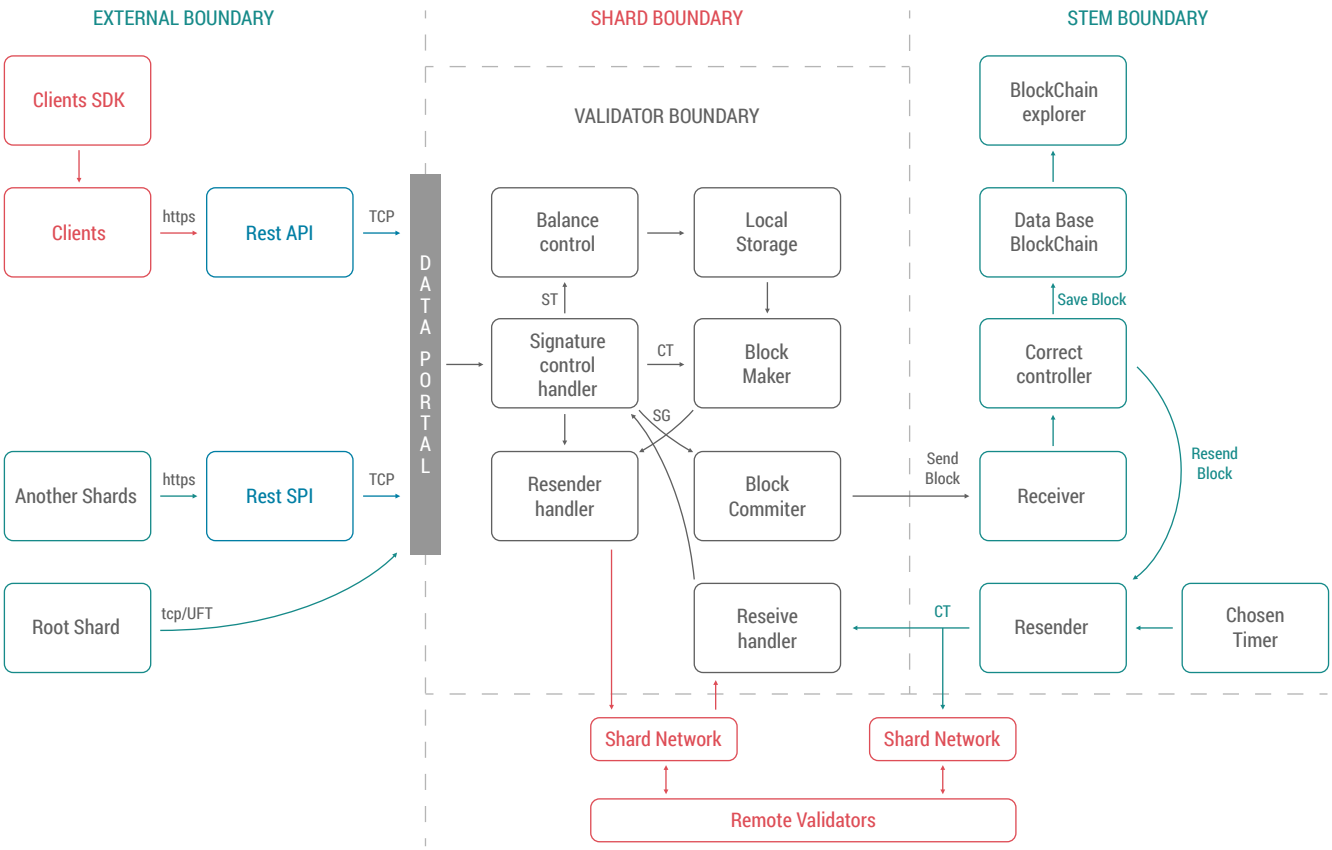
To perform these functions, the Root Shard stores in its blockchain and operates the following types of data:

1. Shard name;
2. Unique shard identifier;
3. Shard type;
4. Access modifiers (public / private / secure);
5. Shard activity sign;
6. The structure of Merkle Trees for every blockchain of every shard.
   The Root Shard does not store copies of all blocks: it uses a Merkle Proof for data validation;
7. A description of the service provided by a shard and its API.

The Root Shard supports the following types of transactions:

1. New shard registration transaction;
2. Shard deletion transaction;
3. Shard block Merkle Tree saving transaction;
4. The transaction of saving the Twig Nodes for a current shard validation session.

5.5 NZT shard



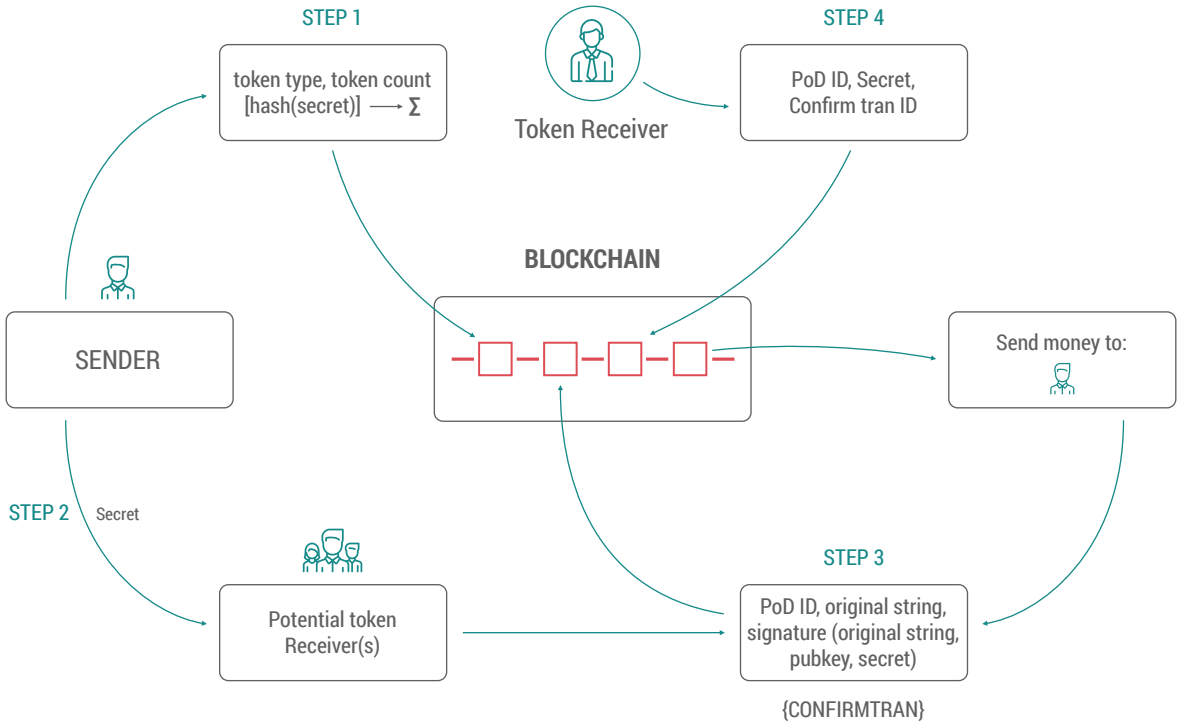
The NZT shard provides the service of work with the main coin of the system, NZT. This is the function of transfer of funds between system users and support for a number of special transactions:

1. A transaction for filing an application for participating in a current validation session as a Twig node;
2. On-demand transactions (PoD);
3. A transaction for transferring funds;
4. A transaction for transferring funds with Multi Signature support.

.

5.6 PoD - Payment on Demand

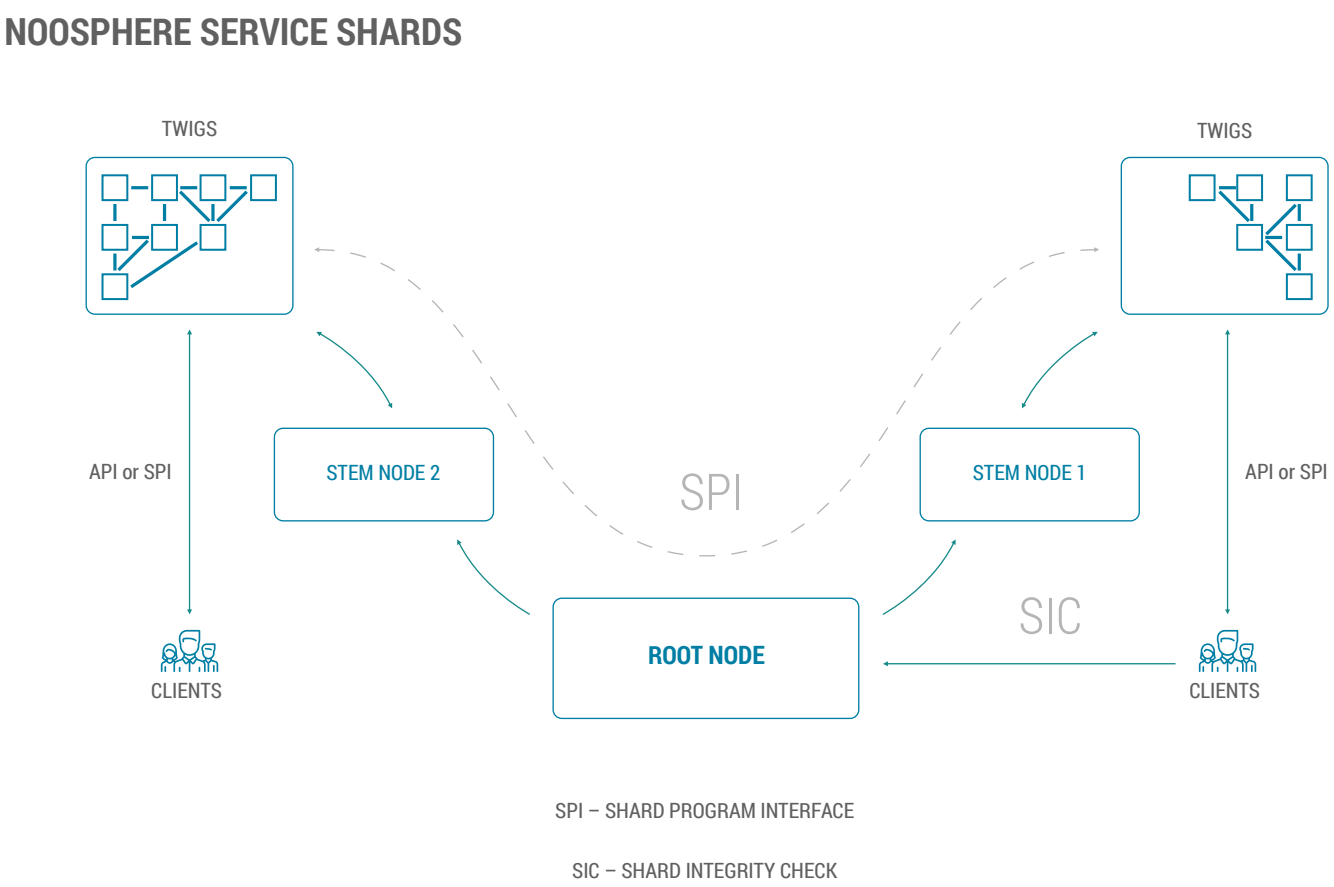
NOOSPHERE PoD – payment on demand



This type of transaction is implemented in the NZT shard and provides the functionality of transfer of funds between users with the participation repudiation feature. A user can switch any number of coins to a special “deposited” state from which they can be withdrawn only after a secret is presented. The secret may be presented by any network participant because coins in a “deposited” state are linked neither to a sender nor to a possible recipient of funds. The transaction creator can generate the necessary number of secrets to divide the amount between them in the necessary proportions.

5.7 Service shards

The ideology of service-oriented sharding suggests switching from the concept of shards to service shards. Every shard in Noosphere is a service shard and these words are considered interchangeable. In the classical implementation of sharding, every shard processes a share of transactions from a total flow, speeding up the blockchain operation. For Noosphere this is only one of usage options because distribution of the load of the inflow of transactions is a service that must be provided by a service shard specially created for that purpose.



There are two ways to implement services at shards. The first method is to use the shard blockchain structure proper in which records of the key-value type are stored according to certain rules and it is these records that determine the functionality of the service. A typical example of this implementation is the DNS, routing, or certifying center services. The second method is more comprehensive and provides services that are more resource-consuming. Their operation requires computational work the result of which will actually determine the content of the blockchain (smart contracts), or else the blockchain will perform exclusively administrative functions if the output data are not meant for storage in the blockchain (rendering, scientific research, dApps). In both cases, every shard, as it is introduced to the system, must provide a description of its API in order to be used.

When a service shard is created, a token associated with it can also be created. Rules must be established for its generation when the shard works at the moment of registration at the Root Shard.

5.8 Shard forks

Noosphere is free of the concept of forks. For a usual blockchain, a fork is a change of the built-in operation algorithms that are critical to the entire system. Thanks to its modular structure, Noosphere enables expanding its functionality by means of creating new service shards. The end user is free to choose the services they trust, so parallel existence of two or more service shards that have the same functionality but different implementation (for example, transaction mixers) is the standard scheme of Noosphere operation.

5.9 Shard security

If a Stem Node participates in the operation of a shard, it is the former that controls selection of participants for a validation session and deletion of untrustworthy validators. A Stem Node participates only in auxiliary functions and the shard blockchain itself is formed without its participation. Therefore, the Stem Node does not influence data security. If shard operation uses the PoET algorithm, all participants being on equal terms, so a consensus among them is reached independently. If network participants find out an untrustworthy Twig Node that in a current validation session has confirmed an invalid block, this Twig Node is blocked by the participants themselves. When shards exchange data, the participants have an opportunity to verify the validity of received data by means of the Root Shard that stores data about all shards. Owning the Merkle Tree of all blocks of all shards, the Root Shard can with 100% confidence verify and confirm the presence and validity of data. Furthermore, the Root Shard stores data about all active and valid nodes of shards that must serve for data requests. If any shard participant is found to present invalid data, the Root Shard can check this and delete the participant.

5.10 Shard Designer

To simplify the creation of one's own service shards, Noosphere provides a special programming tool called Shard Designer. By means of a user-friendly GUI or configuration files, it enables forming the necessary structure of shard transactions and blocks, design the external API of the service and mechanisms of communication with computing resources (including external operating systems). Besides, one can set parameters of the operation of consensus algorithms, if non-standard timeouts are required. Built-in tests will enable avoiding designing errors, and a test network will offer an opportunity to test a new shard in operation and in the environment of other shards.

5.11 Noosphere testnet

The Noosphere testnet consists of two elements. The first is a purely virtual network supplied as part of the Shard Designer and emulating the operation of the Noosphere network on a single computer. The second is a live test network created and supported by the Noosphere Foundation for full-scale testing of new shards, especially when candidates for static shards appear.

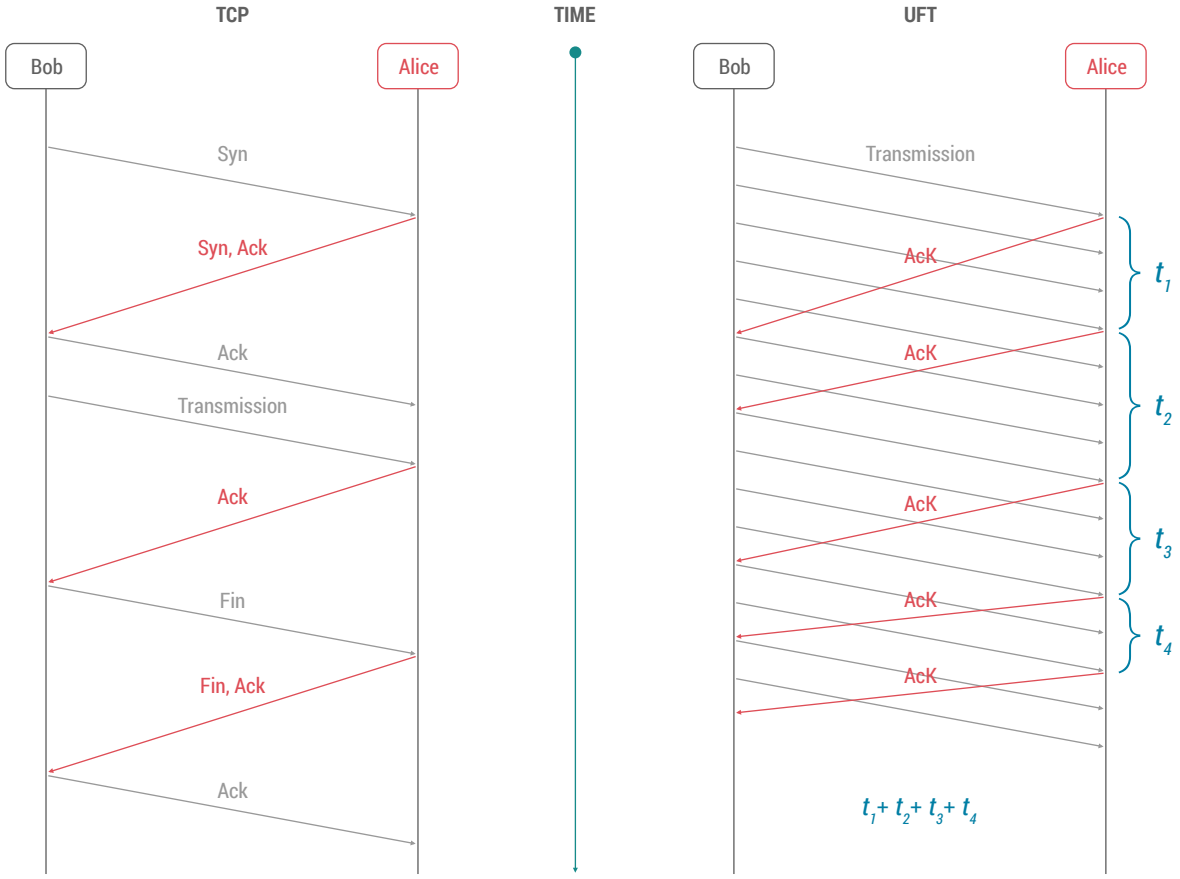
5.12 Parallel transactions execution

The high speed of the operation of Noosphere lays down special requirements for algorithms processing incoming transactions. To avoid queues, parallel processing of the input flow is used. Transactions are distributed between parallel service flows depending on their type and a special value of the flow label obtained from the sender's address. When a validator is triggered, the software determines the possible number of system flows that can be used to service transactions. Depending on that number, an appropriate number of processing pools with unique identifiers are formed to which incoming transactions will be directed. There can be several options for obtaining a flow label from the sender's address. The simplest is division of the address space by the number of flows. Compared with sequential processing, this yields a significant increase in performance, but continues to protect against double spending.

5.13 UFT – UDP Flow Transmission

The UFT (UDP Flow Transmission) protocol underlying the transmission and retransmission of data between shard participants (and directly between shards) enables avoiding problems typical of TCP (which is the de facto standard for data transmission on the Internet): slower transmission speed due to loss of packets, congestion (when the client cannot receive all data the sender has sent). Each of these problems leads to a situation when an operation such as sending transactions for validation may significantly decrease the network speed because of utilizing the network resources of each node for additional, in the case of blockchain, unnecessary overhead.

NOOSPHERE UFT



A decline in the transmission speed in the case of TCP arises because of possible congestion of backbone routers when message queue is congested. As the number of routers in the chain on the network grows, the probability of loss increases, which even in the case of a single loss leads to a quite noticeable decline in the transmission speed. The TCP protocol suggests mutual confirmation of each packet in the transmitted flow, so if the sender does not receive the acknowledgement signal (TCP ACK), these leads to re-initialization of the transmission from the point where the packet was lost. UFT suggests establishment of a two-sided connection between sender and receiver where the sender-receiver connection is used for the sent data flow and the reverse, independent, one is necessary for receipt acknowledgement. The fundamental difference between TCP and UFT is that UFT requires selective acknowledgement of the received flow (SACK), after short time intervals. Periodic ACKs help simplify managing the backward movement of control messages when the data transmission speed is high because in these situations the number of ACKs is proportional to time, not the number of data packets as in the case of TCP. If a packet is lost, the sender sends a negative response (negative ACK, NAK), which leads to selective resending of a specific part of the sent flow.

One mechanism used in TCP to prevent congestion, the so-called “Slow start and additive increase” makes it impossible to send small packets containing transactions so as to use maximum network throughput. When transmission begins using the TCP protocol, the speed is set at a minimum with further increase as the process goes, but this approach totally fails to achieve the goal because transmission ends at a moment when its speed has not yet reached maximum. UFT, on the other hand, uses a channel width control mechanism based on decreasing the data transmission bandwidth proceeding from optimum values sent by the ACK and NAK sender.

Thus replacement of the standard data transmission protocol with UFT enables maximum efficiency in the use of network bandwidth, which leads to faster exchange of information between nodes and faster block validation, and this has a significant effect on the total number of processed transactions and first of all on the speed of exchange between shards.

5.14 Consensus

5.14.1 CBFT consensus

Convolutional Byzantine Fault Tolerance is the main consensus algorithm in Noosphere. Unlike popular PoS, CBFT does not divide system nodes according to the size of a certain contribution. They are all equal and there is just a set of criteria used for their selection:

- computing power;
- internet connection quality;
- having enough coins for the deposit.

The system uses the deposit to guarantee the trustworthiness of a node. Nodes are randomly selected at fixed intervals from among those having filed an application. To file an application, a special type of transaction is used for transparency of the process, and data about all candidates and Twig Nodes are stored in the blockchain. The number of Twig Nodes operating simultaneously during a current validation session depends on network load. The duration of a validation session is 10,000 blocks.

A distinctive feature of the CBFT implementation is the absence of the need for sending out a candidate block to the entire network of Twig Nodes. Instead, to reach a consensus and hold a vote it is enough for nodes to exchange a compressed set of metadata about transactions. Sending of block is only used when new validators appear that need a current copy of the blockchain. In addition, the use of the UFT protocol enables avoiding forced delays in transmission the way it happens in the classical implementations of the BFT algorithm. The beginning of the process of the formation of a current block is controlled by a Stem Node that randomly selects a current Twig Node from among active Twig Nodes. A block takes 1.5 seconds to form, and only one validator has the right to form a block at a given moment of time. If no block is formed after an interval of 1.5 seconds, the Stem Node transfers control to a next Twig Node and doubles the interval. If a validator fails to form a block over an allotted period of time more than 3 times in a row, it is removed from the current set of validators. If a block is formed with false transactions, the validator is deleted and the system withholds their deposit.

5.14.2 CBFT Technical

The network architecture of the consensus systems of existing blockchain platforms is based on the principles that the system has a stationary operation period and is built exclusively with the use of a Poisson flow of events. The need to analyze the form of the input data flow distribution law and the duration of transition periods is in many cases determined by the fact that the latter may make up a significant proportion of the system operation period, while the input flow distribution law may have a significant influence on the statistical characteristics of the output parameters of the system. Therefore, without taking account of the non-stationary period and the influence of the form of the input flow distribution law, it is impossible to optimize the operating characteristics of the system as a whole.

Consider a model of the input flow of jobs in the system that is determined by dependence of the input flow rate on time $\lambda_g(t)$ $(g=1, period)$, given the average rate λ, i.e. the input data are comprised of the value of the average input flow rate λ and statistical information reflecting change in the rate of the input flow of jobs over a certain period of time (*period* – the system operating period). Queueing theory generally defines that jobs arrive in the system exponentially and the rate of the input flow of jobs λ, does not depend on time t. The time interval between incoming jobs ξ is a continuous random value having exponential distribution with the parameter $\lambda > 0$. ξ has only non-negative values, and its density $f(x)$ and distribution function $f(x)$ are as follows, respectively:

$$f_\xi(x) = \begin{cases} \lambda^{-\lambda x}, x \geq 0, \\ 0, x < 0; \end{cases}$$

$$F_\xi(x) = \begin{cases} \lambda^{-\lambda x} 0, x \leq 0, \\ 0, x < 0; \end{cases}$$

Where λ is the input flow rate.

The mathematical expectation of the random value ξ is connected with the input flow rate by the $M_\xi = \frac{1}{\lambda}$. The dispersion of the random value ξ is also determined by the input flow rate $D_\xi = \frac{1}{\lambda^2}$. In practice, the system input flow rate is not a constant value, it changes in time. These changes are connected with the period of sending out a candidate block as consensus is being reached between active system participants and with the service time in the system, namely with the analysis of the candidate block which consists of a set of transactions for the current validation period.

As the CBFT consensus algorithm was being developed, statistical information was collected reflecting change of the input data flow rate in the distributed network depending on the time it took to distribute a candidate block, its size, and processing time. The approximation of the obtained statistics is carried out with the help of the piecewise-polynomial interpolating cubic spline $S_g(t)$ with the setting of boundary conditions, i.e. for each section $[t_j, t_{j+1}]$ with the number j the approximation function $S_g(t)$ has the formof a polynomial.

$$P_j(t) = \sum_{i=0}^{k-1} a_i^{(j)} (t - t_j)^i, k - 1 = 3$$

The boundary conditions consist in the periodicity condition, i.e. coincidence of the values of the first and second derivatives on the borders of the interval $[T_1, T_n]$. The plotting of the spline can be reduced to determining the multiple ratios $a_i^{(j)}$ by solving systems of linear equations. The interpolating spline $S_g(t)$ is plotted so as to satisfy the interpolation condition $S_g(t_i) = y_g(t_i)$ $i = 1, ..., N$ for the table function Y_g.

To obtain the specific dependence of the input flow rate on time $_g(t)$, given the average rate λ, it is necessary to multiply the obtained spline $S_g(t)$ by the average rate and divide it by the average value of the spline $S_g(t)$ itself:

$$\lambda_g(t) = \frac{\lambda}{\overline{S_g(t)}} S_g(t)$$

$$\overline{S_g}(t) = \frac{\sum_{i=1}^{N} S_g(t_i)}{N}$$

where λ is the actual average input flow rate.

The proposed input flow model suggests that the time between incoming jobs ξ is a continuous random value that may be distributed according to different distribution laws, such as the exponential law, Poisson law, normal law, and uniform law.

5.14.3 Proof-of-elapsed-time (PoET)

If the governing node fails in any shard, Noosphere suggests for it a switch to another consensus algorithm, PoET. This algorithm works as follows. Every node of a shard initiates an internal timer counting random time. The first node whose timer goes off earlier than the others becomes the validator of the current block. However, to implement this functionality two conditions have to be checked: how random the waiting time was and whether the node indeed waited for a timer to go off. Checking these conditions becomes possible with the use of the Intel Software Guard Extensions (SGX) technology. SGX enables applications to run trusted code in a protected environment at the processor level. For the PoET consensus algorithm, a code is trusted when it guarantees that the above-mentioned two conditions are met. SGX enables network participants to check each other for whether a correct code has been loaded into the protected environment and whether it is run correctly, which excludes the opportunity of manipulation for a wrongdoer. The time for reaching consensus is extended by 4 seconds in this case compared with the key algorithm, CBFT. This consensus algorithm can also be used if a shard has no increased speed requirements for transaction processing or if no Stem Node is required for its operation.

5.14.4 Hybrid SGX-CBFT algorithm

For critical services that may be expected to be under attack with a high probability, a user may use a hybrid consensus protocol, SGX-CBFT. It works in the same way as the above-described Convolutional BFT, but network participants get an additional opportunity of remote reliability verification of all neighboring nodes thanks to the Intel SGX technology.

5.15 Attacks protection

Blockchain, like any internet-based information system, is a target for hackers. Besides the known type of attack, DDoS, new methods appear for affecting specifically blockchain platforms. Analysis of all these threats, including research into the shortcomings of the architecture of existing blockchain systems that have been subjected to attacks, has made it possible to build effective methods for proactive defense into the Noosphere subsystems.

5.15.1 DDoS – Distributed Denial of Service

The success of this attack is based on limiting throughput which is one of the characteristics of any network-based resource. A large number of requests are sent to a resource under DDoS attack with the aim of exhausting its potential for data processing and disrupting its normal operation. A special service shard called Loki may be used to prevent this threat. One of its tasks is constant monitoring and analysis of data, so when there are signs of a DDoS attack, the traffic is routed to special nodes that imitate processing. As a result, the attacker gets the impression of a successful attack, while the protected service continues normal operation.

5.15.2 Sybil attack

This is an attack when a wrongdoer fills the network with numerous controlled nodes trying to “surround” the victim’s node, i.e. get control of all neighboring network nodes. This type is inapplicable to the CBFT consensus algorithm because any attempt at disrupting normal operation of the network will result in blocking the node, the system withholding its deposit. Even if the wrongdoer has enough resources to resume the attack, nodes are selected randomly for every round of validation, which significantly decreases the probability of success.

5.15.3 Eclipse Attack

This kind of cyber-attack was described in detail in a 2015 report by a group of scientists from the Boston and Hebrew universities led by Ethan Heilman and later demonstrated using the Ethereum network as an example. Using correct manipulation in a peer-to-peer network, a hacker can "obscure" nodes in such a way as to make them contact only infected nodes and thereby influence the process of forming new blocks. For the consensus algorithms used in Noosphere, this attack is unrealizable. Owing to the Intel SGX technology, the PoET algorithm enables reliability verification of neighboring nodes, while CBFT, due to random change of validators in every validation round, forces the attacker to start the attack afresh every time, which negates its effectiveness.

5.15.4 51% Attack

This attack is among the most popular and has been carried out against mining-based blockchains many times. For other systems, not based on mining, including for Noosphere, this attack becomes difficult to carry out because it requires orders of magnitude more economic resources. If the SGX-CBFT hybrid is used, the attack becomes altogether impossible because even if one node is captured, it will instantly be localized and blocked thanks to the possibility of remote reliability verification.

5.15.5 Double-spending Attack

This attack suggests that the same funds are successfully used more than once if blockchain goes out of sync. This attack is only typical of systems in which every network participant has the right to form new blocks. For the CBFT consensus algorithm this attack is unrealizable because only one node is responsible for forming a block over a specific period of time and the network cannot go out of sync.

5.16 Swift Torus Routing

Modern internet routing protocols were created to be used in centralized systems. Existing decentralized platforms use them because there are no analogues and alternatives. These protocols are nonetheless not an efficient solution to the problems of routing in decentralized systems because they operate absolutely different entities and the need for fast routing between dynamic nodes is several times higher.

Decentralized networks include a large number of transit nodes, orders of magnitude more than centralized. Solving the problem of routing on such networks not only requires mathematics that is more complex, but the very calculation of routes becomes impossible without creating special software. To solve these problems, a protocol called Swift Torus Routing is being developed. Its use enables considerably increasing the efficiency and speed of data transmission between dynamic nodes, namely when transmitting data between nodes of different shards in the absence of a centralized point of entry into the system implemented by a service shard.

One method for building dynamic routes in systems with a dynamic network topology is the use of graphs whose points are network nodes and connections between them are weighted communication channels. Instead of standard graph traversing algorithms, the calculation algorithm used to find the shortest path is based on space-efficient graph representation – Hilbert-ordered tiles. This method is used in a modern graph processing engine named MOSAIC. For comparison, 4 Xeon Phi processors using the Hilbert-ordered tiling scheme take one second to calculate routes based on a graph of approximately 700,000,000 points.

Discrete time models are used to form the network graph proper. These types of models represent network topology changes as the periodically recurring sequence S of topology snapshots divided by the duration interval $\Delta t = T/S$, where T is the recurrence period of the topological states of the set.

Each snapshot is matched to the graph G = (V, E) where V is the set of nodes and E the set of communication channels. For the finite set of graphs {G} recurring over the period T, routing table are pre-calculated. These tables are distributed between nodes and used when necessary. Δt

The mathematical model of the system makes it possible to present continuous topological changes in the form of a sequence of steady states described with the help of graphs:

$$\forall t \in [0; +\infty) \exists \Delta t_n = t_n - t_{n-1}$$
$$t \in (t_{n-1}; t_n)$$
$$G_n(t) = G(\Delta t_n)$$

to take into account that the topological state of the network changes in time under the impact of external events:

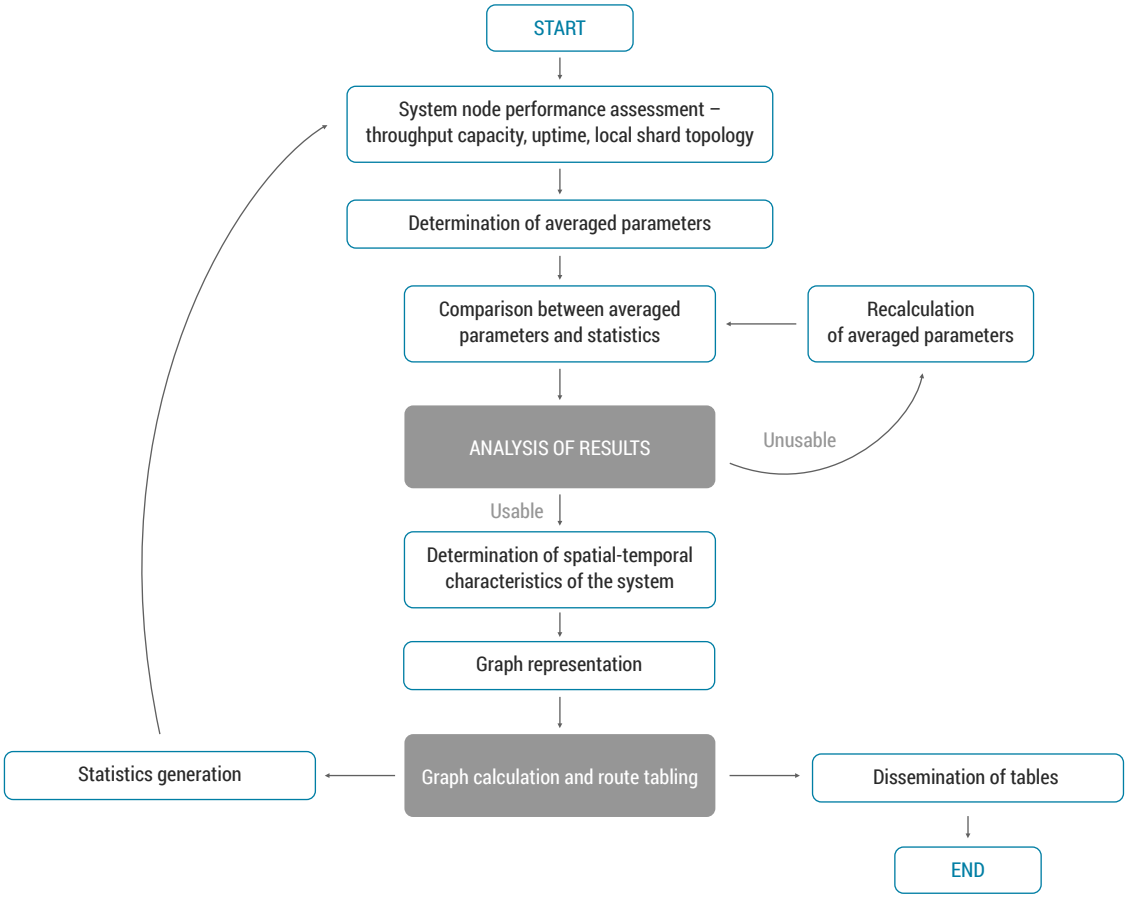
$$G(\Delta t_0) \rightarrow G(\Delta t_1) \rightarrow \ldots \rightarrow G(\Delta t_n)$$

to forecast the future state of the network topology for a certain time – the forecast time.

$$\sum_{1}^{n} \Delta t_n = T_p$$

where Δt is the time during which the system parameters are considered not to worsen; $G(\Delta t_n)$ is the network graph at the moment with a finite set of points V and a set of edges E. T_p is the time of the forecast state of the communication system.

Considering the above-stated criteria, the key input parameters of the model are statistics on the operation of nodes and channels and performance characteristics of equipment. Below is a flow chart of the algorithm of pre-calculation of routing tables.



5.17 Noosphere Service-Shards Applications

The Noosphere Foundation community of developers will create the main business-critical services based on the principles of decentralization and security. Their task is to serve as the basis for the operation of services of greater complexity that stakeholders will develop.

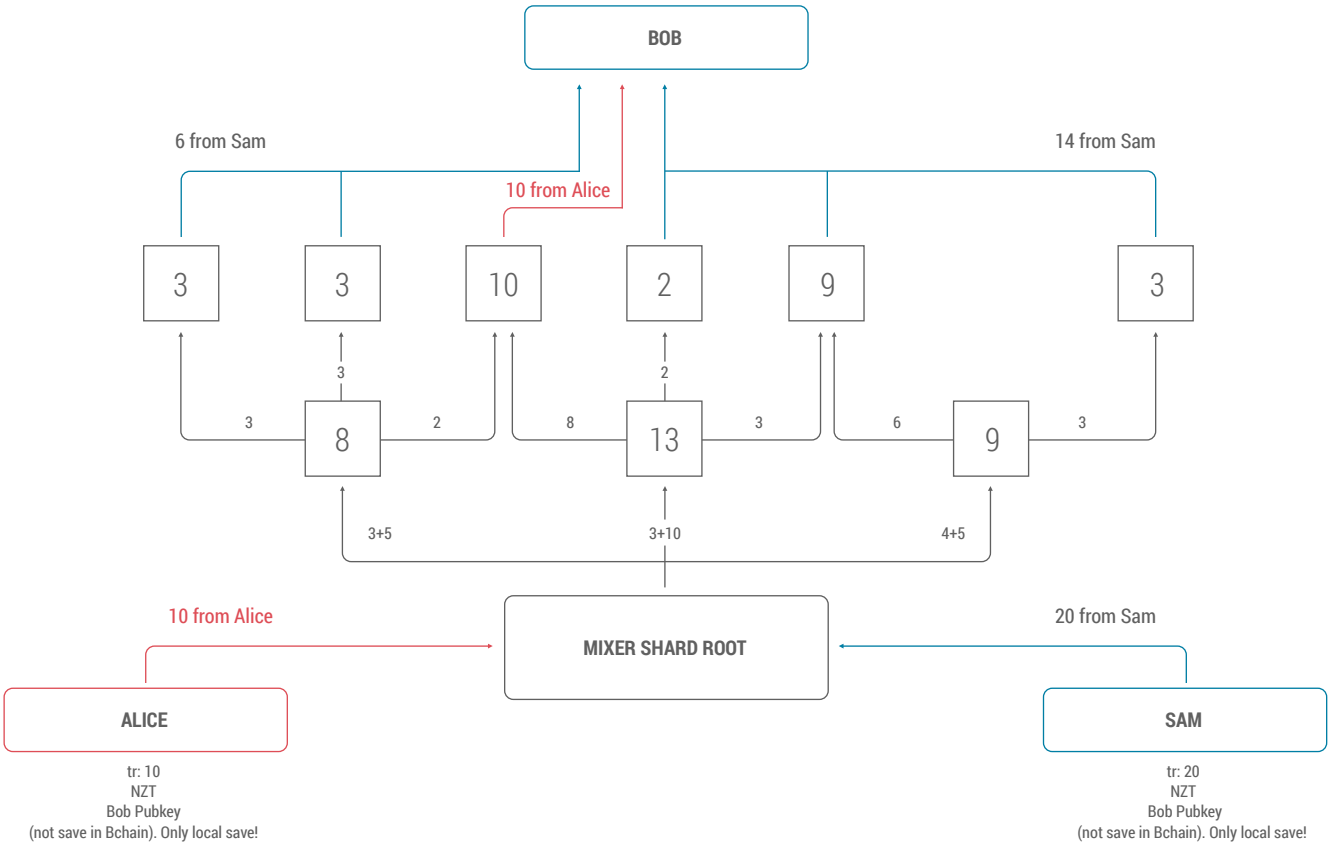
5.17.1 DAR & DDNS – Dynamic Application Routing & Dynamic Domain Name System

One possible option for using service-oriented sharding is the possibility of using the computing resources of a network as a distributed data store or distributed router. Such a router can be implemented with the use of the Dynamic Application Routing protocol the Noosphere Foundation has developed. This protocol suggests the use of the throughput potential of each network participant for creating a decentralized network over the internet modeled on the I2P or Tor networks, yet not at the level of general transmission of IP packets, but at the level of data exchange between client and server using the protocol of a specific application. The implementation of this protocol also enables one to use modern algorithms for balancing load between end servers as well as routing user data between geo-distributed data centers. Using the implementation of the protocol can serve as a method of random routing of packets between server and client, applicable to Noosphere Secret Messenger.

Every node of a shard is a routing unit. The information about its IP address and connection characteristics (response time and throughput) is stored in the specific blockchain shard. To implement routing with geo-distribution, the blockchain stores a key-value combination where key is the hash of a user identifier and value is the necessary point of entry into the system. Thus every network participant will have specific instructions as to which server the data of a specific user must be directed to. The question arises how a user of the messenger who is not connected with the blockchain shard will decide which router to get connected to. The decision is quite trivial. The DNS protocol enables updating information on a zone once in 5 seconds. The real time of updating the global record is about 25 minutes, which enables designing this application shard so as to form a block containing information about routers once in 5-10 minutes. During this time, information is formed about the DNS zone from blockchain and uploaded for global processing and distribution inside the DNS peering network. Thus the messenger client establishing connection requests the current IP address of the router in the form of a DNS request and the DNS server having written updated data from blockchain into its cache gives one of the addresses of the shard participants. The client, upon receiving the router address, connects to the server in their region and begins data exchange. The main advantage of this approach is complete impossibility of blocking a service whose data communication is based on dynamic application routing because incoming traffic from messenger clients has the form of requests and responses over the standard https protocol with constantly changing points of entry.

5.17.2 NTM – Noosphere Transaction Mixer

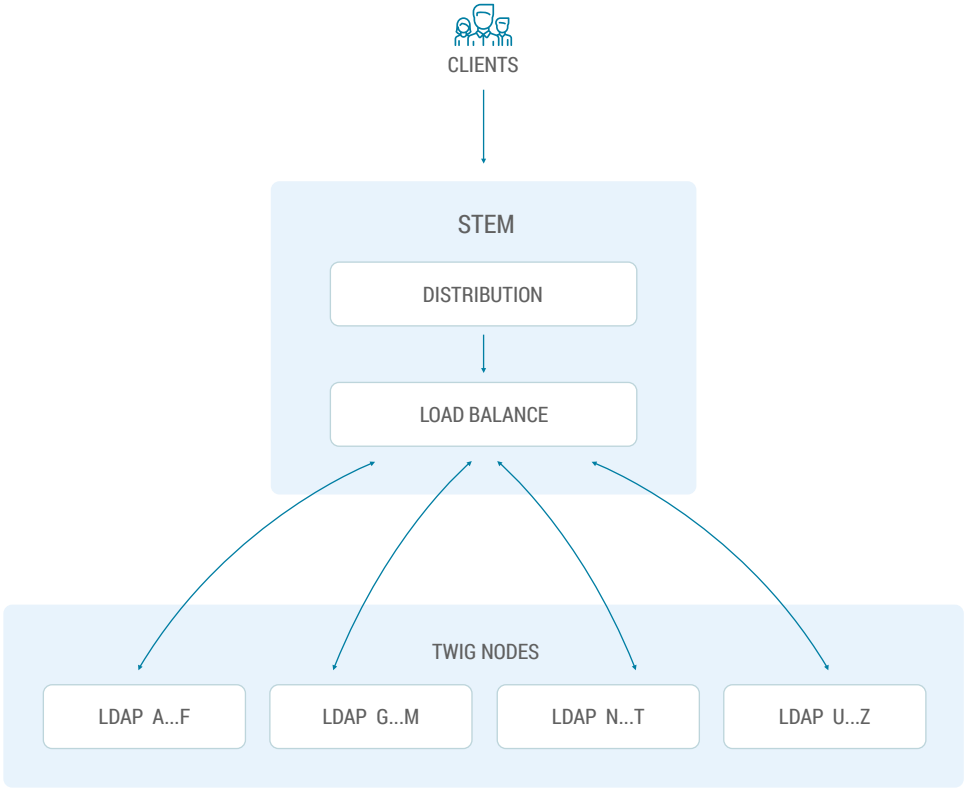
NOOSPHERE NTM



The transaction mixer popular among blockchain users for increasing the anonymity of transfers can easily be implemented in Noosphere. A user transfers tokens to the Stem Node of a shard indicating the end recipient of the funds. The Stem Node divides the amount into numerous smaller ones and sends them to Twig Nodes with an end recipient label. Network nodes exchange these tokens during a user-selected time and send the funds to the recipient at random moments. The shard Stem Node maintains control to ensure that all funds are transferred. The shard will store no information about senders and recipients. Because several mixer options can be implemented simultaneously by different users, it is easy to use these mixers one after another, which greatly improves the anonymity of transfers keeping users' personal data secret.

5.17.3 DDAP – Decentralized Directory Access Protocol

NOOSPHERE DDAP



DDAP is a protocol providing the functionality of the LDAP protocol with higher requirements for security, fault tolerance, and operating speed.
All stored catalogues are distributed between the nodes of a shard, and the reservation degree of each type of data can be set separately. The shard Stem Node determines the shard node to which a user request will be redirected, allowing for the current load on the shard and the list of shards possessing the requested information.
If one or several shard nodes malfunction, all requests will be redirected to other nodes, transparently for the user.

5.17.4 ACS – Autonomous Copyright System

The architecture of blockchain systems enables creating autonomous copyright systems protected against destruction and modification. Any data may be copyrighted that can be presented in binary code: images, text, audio, and others. The ACS service creates for any object a number of attributes that serve as its unique identifier and enables a quick search of the entire database. A certificate can be obtained for each object from the database that can help protect you against copyright infringement.

5.17.5 Loki – DDoS Protection System

DDoS (Distributed Denial of Service) – a distributed attack of the "denial of service" type. A network resource's operation is disrupted due to a flood of requests being sent to it from many different sources.

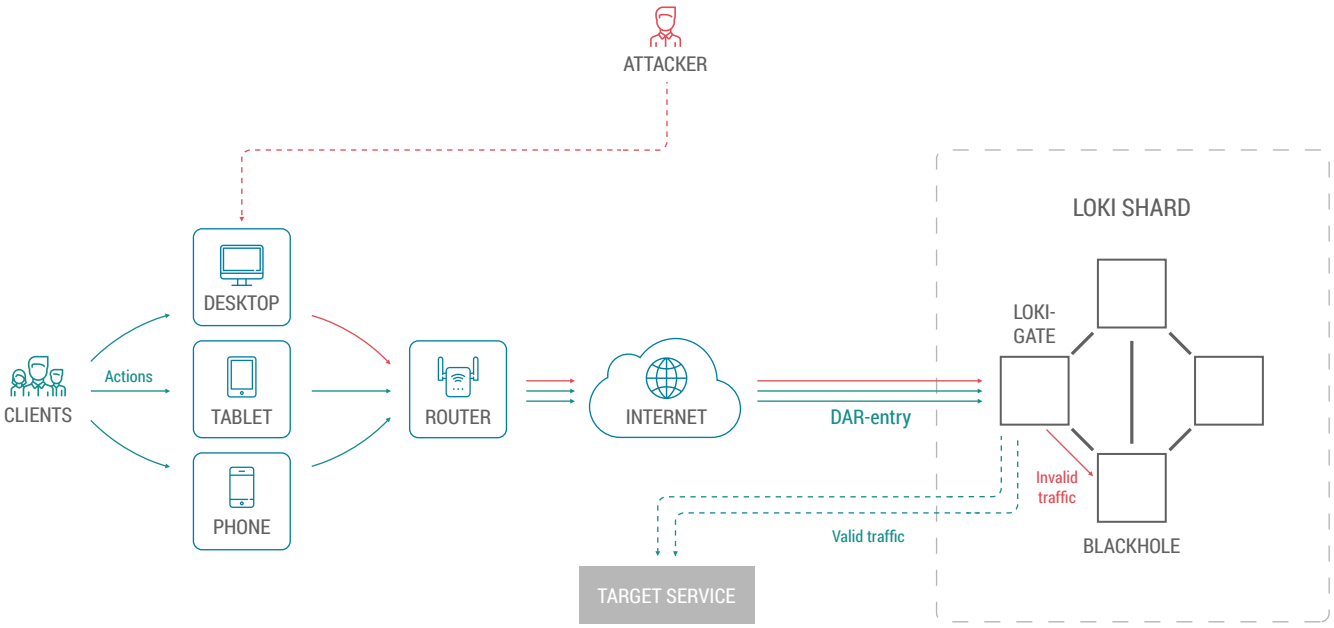
There can be a huge number of reasons for such attacks, and building lines of defense does not require going into detail as to why an attack is carried out. Also, there is quite a broad range of methods for carrying out such an attack, just like ways to be protected against this.

Loki provides a service for analyzing incoming requests, communication sessions, and data flows, working simultaneously at several levels of the OSI model, combining them to identify traffic coming from a legitimate user with maximum accuracy, and reducing the probability of false positives to a zero.

There are several basic techniques used by Loki:

- Filtering out malformed data flow
- Analyzing the data flow and identifying anomalies therein
- Analyzing user activities on a site

NOOSPHERE LOKI



While the first technique contains a set of firewall filtering rules, the second and third suggest the use of DPI (deep packet inspection) for collecting statistics and carrying out machine education based on the statistics for further identification of anomalies and quick response to them.

In this case, blockchain is used to store weights of the convolutional neural network for identifying traffic anomalies, which enables synchronizing the results of educating the CNN between the filtering nodes.

If an anomaly is discovered in a data flow or the statistics on user activities or if there are failures in setting flow control sequences, the user request is automatically redirected for additional verification, which in the first place will help additionally educate the neural network by means of validating and identifying the user, and if an attack is detected with certainty, the traffic is redirected to special nodes simulating the attacker success.

The use of this set of techniques in combination with the DAR technique presented in this paper will make it possible to create a service with maximum fault tolerance, regardless of the nature of the service, whether a messenger service, a web site, a VPN gate, or anything else.

5.17.6 EBS – Effective Backup Service

Using a blockchain to store distributed data is way more advantageous than using a centralized system. The speed of requesting and downloading data is orders of magnitude higher because they data is downloaded simultaneously from many sources sharing the file. The only bottleneck is data integrity and security if a distributed network participant is out of order or compromised. This drawback is easily overcome with the use of the Forward Error Correction (FEC) schemes based on the Vandermonde matrix. The Effective Backup Service provides erasure recovery and error correction features that enable maintaining data integrity and security even if a part of them is lost. There are a number of error correction codes that can be used in FEC. These are Reed-Solomon code, Hamming code, Reed-Muller code, binary Golay code and others.

Each of those codes essentially relies on sending a portion of redundant data necessary for data recovery in the case of errors or losses. The average size of redundant data necessary for a guaranteed backup is 30%. However, unlike centralized storage, 30% is payment for high downloading and uploading speeds that are orders of magnitude higher than those offered by any existing services. This is achieved both by distributing data between numerous nodes and through an absence of the need for resending data in the cases of losses in the network. Data security in the case of using a decentralized store also increases compared with a centralized one because only the data owner knows which nodes share their data and only they know the encryption codes of each chunk of data. The preservation of data in nodes is additionally guaranteed by a deposit distributed between all EBS users whose data were stored in a malfunctioning node.

5.17.7 DHPC – Decentralized High Performance Computing

One method for implementing distributed computing is Grid systems. A grid is a geographically distributed computing platform consisting of heterogeneous nodes accessed via a single interface. It co-ordinates the use of resources when there is no centralized management of these resources; uses standard, open, universal protocols and interfaces; and nontrivially ensures high-quality service. Grid systems are divided into several types of resources:

- Data grid;
- Information grid;
- Computing grid.

Grid applications include:

- Complex modeling on remote supercomputers;
- Joint visualization of very large sets of scientific data;
- Distributed processing for data analysis;
- Video rendering.

Leading grid solutions are based on the Open Grid Service Architecture (OGSA). OGSA is a distributed computation and interaction architecture based on services and guaranteeing interoperability of heterogeneous systems regardless of their implementation, location, and platforms in such a manner as to enable different types of resources to communicate and exchange information.

Noosphere develops a DHPC service based on OGSA with the aim of efficient use of computing resources both in existing projects like HTCondor, Globus Project and BOINC and in new ones, based on OGSA.

DHPC includes:

- The client part of the distributed application, serving as an integrated portal to all distributed grid computing platforms;
- An SDK based on HTCondor for performing computing tasks;
- A control point mechanism;
- A data replication service.

DHPC will make it possible both to rent one's computing resources and to use the computing resources of grid platforms for performing one's one tasks.

5.17.8 PVM – Python Virtual Machine

Noosphere imposes no restrictions on features to be implemented in its service shards. This means that no restrictions are imposed on the types of virtual machines to be used in shards for implementing the functionality of smart contracts and dApps. The Noosphere Foundation implements a new type of virtual machine for processing smart contracts that is based on a Python-like language. Python programming language has been chosen as the basis because Github statistics show that it ranks first in popularity among programming languages for quick and efficient creation of scripts which in our case are smart contracts. This service shard is not the main shard for processing smart contracts in Noosphere, but only provides this functionality to everyone, along with other similar service shards.

5.17.9 ABG – Any Blockchain Gate

Any blockchain is based on unified abstract functions: work with transactions and API for requesting data from root nodes. Internal implementation may vary, but the access interface can be uniform. A template service, ABG, is used for this purpose which enables introducing any third-party blockchains to the Noosphere ecosystem. End users will be provided with a uniform access API enabling flexible use of any systems to create smart contracts and dApps that use multiple data sources in their operation.

6 Conclusion

The Noosphere platform was designed on the basis of experience of comprehensive research into advanced blockchain technologies, cryptography, and cybernetics. Thanks to its service-oriented approach, Noosphere creates the first blockchain ecosystem to become a new step in the development of information technology and to improve the life of every man on Earth.