The object of this study is the protocol for detecting nodes in the Rootstock blockchain network and crawling tools. Node discovery protocols are the foundation of any decentralized peer-to-peer network. In blockchain systems, full nodes store and maintain a complete copy of all transactions performed in the network. However, they do not store information about all other nodes, such as their IDs or IP addresses. Each node usually maintains an incomplete list of nodes to which it connects to exchange blockchain data. In decentralized networks, nodes join and leave the network and their IP addresses can change, making it impractical to maintain a complete, up-to-date list of all nodes. Therefore, the only way to get a list of all nodes in the network is to poll each node sequentially.

The developed method involves sending specially formed messages to nodes to obtain their neighbors. The graph search algorithm is used to traverse all received neighboring nodes. This allows one to consistently detect all network nodes. Identifying the desired sequence of messages requires a preliminary analysis of the node software RSKj in the part of node discovery protocol.

Effectiveness of the proposed method was verified using the developed software and an experiment in the main network. 6 verification nodes were deployed in different physical locations and at different times. All test nodes were detected in less than 10 minutes. The developed method found 222 nodes that have 209 unique IP addresses.

Results of this study show how to perform analysis of node discovery protocol. They provide the means to obtain information about the available nodes of the Rootstock blockchain system, enabling the analysis of both the blockchain network in general and individual node

Keywords: peer discovery, network crawling, peer-to-peer networks, Rootstock blockchain, Kademlia, decentralization

UDC 004.4+004.9
DOI: 10.15587/1729-4061.2024.297903

DEVELOPING A METHOD FOR THE DETECTION AND IDENTIFICATION OF ROOTSTOCK BLOCKCHAIN NETWORK NODES

Yaroslaw Dorogyy
Doctor of Technical Sciences, Associate Professor
Department of Information Systems and Technologies*
Vadym Kolisnichenko
Corresponding author
Postgraduate Student
Department of Computer Science and Software Engineering*
E-mail: vadym.kolisnichenko@gmail.com
*National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Beresteiskyi (Peremohy) ave., 37, Kyiv, Ukraine, 03056

Received date 17.11.2023
Accepted date 29.01.2024
Published date 28.02.2024

***How to Cite**: Dorogyy, Y., Kolisnichenko, V. (2024). Developing a method for the detection and identification of Rootstock blockchain network nodes. Eastern-European Journal of Enterprise Technologies, 1 (2 (127)), 6–15. doi: https://doi.org/10.15587/1729-4061.2024.297903*

1. Introduction

Blockchain nodes are one of the key elements of the network. They receive, verify, and broadcast transactions and blocks. Although nodes in blockchain networks are equal, some of them can perform special functions [1]. One such type of node is the boot node, which is the first node to which a new node connects. They usually do not implement the underlying blockchain logic but only serve to provide information about network nodes for further connection. Another special type of nodes is mining nodes. These nodes implement all the same logic as normal nodes but in addition, with the help of the mining process, they generate new blocks from the received transactions (in some blockchain networks, the mining process is replaced by the staking process [2]). In addition, some blockchain networks, such as Rootstock (RSK) [3], may have additional types of nodes that enable their specific protocol to operate. Thus, RSK has PowPeg nodes that provide two-way communication with the Bitcoin network [4].

Usually, the network interaction of blockchain nodes is implemented by three protocols. The first protocol is Peer Discovery, it is responsible for discovering nodes in the blockchain network and uses the User Datagram Protocol (UDP). The second protocol is Wire, it uses the Transmission Control Protocol (TCP) protocol and implements the exchange of blockchain information between nodes. The third protocol – JSON-RPC is built on the Hypertext Transfer Protocol (HTTP) protocol for the interaction of external applications with the blockchain node.

Peer Discovery and other network protocols do not involve storing, connecting, or providing all network nodes, as the system works in a decentralized manner. Each node that connects to the network actually connects to a subset of nodes, so it is impossible to get a list of all nodes from a single node.

Collecting all network nodes is an important task of blockchain network analysis for both its developers and external analysts. It allows developers to understand the scale of the network and track its dynamics. Analysts need to find specific nodes, or nodes that meet certain criteria (for example, located in a certain region) for further analysis.

Therefore, scientific studies on this topic are important because they show how to analyze node detection protocols for a selected network, determine its limitations, and how to bypass these limitations for further analysis. Our study describes in detail the principles of the protocol for finding nodes in the Rootstock network, the formats and sequence of messages used.

The results of such studies are needed in practice because they provide approaches and means for obtaining all available nodes of a decentralized peer-to-peer (P2P) system

through proper communication with nodes using the Peer Discovery protocol. This, in turn, allows researchers and developers to analyze the blockchain network as a whole, as well as individual nodes.

2. Literature review and problem statement

Paper [5] presents NodeMaps, a framework for collecting, analyzing, and visualizing data from various popular blockchain platforms such as Cosmos, Stellar, Bitcoin, and Lightning Network. The authors compare how these platforms are distributed around the world, what hosts they use, and what software clients they have. They found that Bitcoin and the Lightning Network have a large geographic coverage and many nodes operate through the TOR network, which ensures privacy. Instead, Cosmos and Stellar blockchain nodes are more likely to operate on large cloud platforms or data centers. But this method cannot be applied to the Rootstock network since the node search protocol is different.

In paper [6], the authors explain what they mean by inaccessible network nodes, and then present the Passive Announcement Listening method. This method allows one to estimate the number of unreachable nodes by analyzing messages on the network that report active IP addresses. Using the PAL method, the authors examine data from the Bitcoin network over a period of more than five years (from 2015 to 2020). Unsolved problems of the PAL method include its accuracy in identifying unreachable network members and adaptation to dynamic network conditions. Its application to other blockchain networks is difficult due to the variety of P2P structures and protocols, requiring specific modifications adapted to the unique protocols and characteristics of each network.

Study [7] analyzes the P2P reliability of the Ethereum network with the help of a developed tool for gathering data about nodes. The study found a strong concentration of nodes in several autonomous systems and suspicious patterns that could indicate a Sybil attack. Applying this detection method to other blockchain networks such as Rootstock is difficult. The reason for this is that the crawler requires adaptation to match the unique mechanisms and structural features of each network.

Paper [8] demonstrates that Go Ethereum (Geth), an implementation of Ethereum, is vulnerable to Eclipse-type attacks. The attack uses the Kademlia peering logic used by Geth, allowing for easy isolation of long-lived remote victim nodes with little resources. The authors discuss the fundamental properties of Geth's node discovery logic that allow for a suitable attack. They analyze the node discovery protocol in the Ethereum network, but the results of this analysis cannot be directly used for other blockchain networks.

Paper [9] studies the topology of the Bitcoin network, in which nodes are scattered around the world. The work presents BTCmap – a framework for detecting and building a map of the Bitcoin network. This framework includes two modules: for collecting address databases from each node and a Bitcoin node emulator for neighbor selection and topology generation. Applying the BTCmap framework to other blockchain networks has potential but also faces some challenges. Each blockchain network has its own unique characteristics and architecture, which may require adaptation and modification of the framework for effective use. It is also important to consider the different scales and complexities of networks.

In study [10], authors conducted a detailed data collection of Bitcoin nodes during the year, from July 2021 to June 2022, covering 127,613 nodes. They found that the number of Bitcoin nodes fluctuates in response to fluctuations in the Onion network. The authors also selected 2,694 Bitcoin key nodes and analyzed their important characteristics. As a result, they used machine learning to group these nodes by various attributes and infer centralization in the Bitcoin network. Bitcoin node research has unsolved issues, such as the analysis of inaccessible nodes and the need to improve clustering algorithms and datasets. These challenges highlight the need for continuous improvement in node discovery methods for blockchain networks to better understand their structure and vulnerabilities. In addition, each blockchain network has its own specific characteristics, so analysis and detection methods may require adaptation.

Article [11] describes a new TopoShot method for detecting the topology of a blockchain network. TopoShot can be used with major Ethereum clients such as Geth and Parity. One of the challenges of this research is to extend the methodology for continuous monitoring and analysis of the dynamic topology of the main Ethereum network in real time since the structure of the network and its connections can change rapidly. Additionally, applying this method to other blockchain networks may require significant adaptation due to differences in network protocols and client software.

Work [12] investigates the topology of the Bitcoin network, which is important for the distribution of transactions and blocks. The authors developed AddressProbe, a method that detects connections in the Bitcoin P2P network. The research examines protocol messages for obtaining neighbor nodes. Topology analysis allows one to identify nodes with a high degree of connection, as well as identify influential nodes that are connected to mining pools. Among the unsolved problems of the developed method is the problem of reliability. The method relies on unofficial specifications of the Bitcoin client, which may change in the future. Applying the AddressProbe method to other blockchain networks can be difficult due to the differences in their operation, indicating the need for more flexible methods to understand the structure of different blockchain networks.

Paper [13] describes Kadcast, a new method of distributing blocks in blockchain networks. Kadcast leverages an existing network architecture, Kademlia, for cost-efficient data delivery. The protocol is based on UDP and includes error correction technology for greater reliability while remaining easy to use. Applying the results of this research to node discovery and data collection in blockchain networks may require adaptation to handle different network conditions and different architectures.

Study [14] reports characteristics of the Bitcoin P2P network based on measurements carried out from 2015 to 2018. Network characterization provides information about the behavior of nodes and their operators, for example, providing evidence that the Bitcoin P2P network has experienced Sybil attacks in the past. One of the challenges of research is the uncertainty about the causes of some effects in the network due to the lack of data from remote nodes. Regarding the application of the developed node detection methods to other blockchain networks, adaptations are needed to take into account different network structures and protocols.

Work [15] considers the Ethereum P2P network. The authors analyze the P2P network structure of the Ethereum system for two months in the mainnet, collecting data from more than 217,514 different nodes. The collected data made it possible to analyze the status of nodes, the type of node services, the geographic distribution of nodes and Internet service providers. The study also highlights several outstanding issues, including the need for more detailed analysis of information dissemination, which could be improved by collecting more data and increasing the number of network connections. In addition, there is potential to develop criteria for inclusion of transactions by miners. Applying these detection methods to other blockchain networks will require adaptation to the unique characteristics of each network, which highlights the need to use individualized approaches in analyzing blockchain networks.

In [16], authors propose the Ethna method, which measures the degrees of Ethereum nodes and determines the delay in message propagation in the P2P Ethereum network. Experiments were also carried out in the work to analyze the topological features of the corresponding P2P network. However, issues such as the need for continuous analysis to track network evolution remain overlooked. In addition, node detection methods developed for the Ethereum network cannot be directly applied to other P2P systems.

Work [17] considers how crawling BitTorrent network nodes can be used to rebuild pirated search engines and to monitor user actions. The authors of the paper developed the Vuze DHT scanner, which was used to demonstrate the limitations of the node discovery protocol, including the inefficient provision of user anonymity. Research on node crawling in the BitTorrent network highlights unsolved problems, such as the difficulty in permanently shutting down illegal torrent nodes due to their easy recovery. In addition, it raises questions about user privacy and the effectiveness of legal action against individual users. Applying this BitTorrent network crawling method to other P2P networks can be useful for understanding their activity. However, different architectures and protocols, such as in blockchain systems or decentralized storage systems, may require individual adaptation of these methods.

The general review of the literature [5–17] allows us to state that the issue of analyzing the node search protocol and developing tools for crawling is important for network developers, its participants, and analysts. However, no studies were found that focused on the Rootstock blockchain network. In addition, no information about the state of the network in terms of the number of deployed nodes and their location is revealed. Node discovery protocols are decentralized, so a single node cannot provide information about the entire network. To get a list of all network nodes, you need to poll each node. Although many of the protocols are well known and have a sufficient theoretical basis, their implementation in different systems differs. Available sources do not provide information about the Rootstock network node search protocol.

3. The aim and objectives of the study

The purpose of our study is to develop a method for detecting and identifying nodes in the Rootstock blockchain network. This will make it possible to effectively identify all nodes operating in a given blockchain network, their IP addresses, and public keys. This information can be used for further analysis of both the nodes themselves and for correlation with on-chain data. In addition, it will make it possible to create a "map" of the blockchain network and draw conclusions about its dynamics and state.

To achieve the goal, the following tasks were set:

– to analyze the Peer Discovery protocol of the Rootstock network blockchain;

– to determine the main stages of the node detection method;

– to develop a prototype of node detection software;

– to evaluate the effectiveness of the proposed method.

4. The study materials and methods

The object of our study is the Peer Discovery protocol of the Rootstock blockchain network. The main hypothesis assumes the application of the graph traversal algorithm and the use of the defined specifics of the protocol implementation in the RSKj software [18] for effective search of network nodes.

Rootstock is a sidechain of the Bitcoin network implemented using the process of merged mining. RSK implements the Rootstock Virtual Machine (RVM), which is similar to the Ethereum virtual network, allowing the creation of smart contracts. Other improvements that RSK brings to the Bitcoin ecosystem include faster block generation and lower transaction costs. Today, there is only one software implementation of a network node – RSKj.

This work uses the current version of RSKj at the time of the research – Fingerroot 5.3.0. Analysis of the protocol was performed through the RSKj source code review and through the analysis of the documentation of a similar protocol in the Ethereum blockchain network [19].

The theory of algorithms is used to build the node search method, namely, graph search [20].

The experimental part involves the deployment of a certain number of nodes in the main network (mainnet), which will be used as a check that the software-implemented method managed to find the relevant nodes when searching for all of them. Verification nodes are deployed at a given frequency on the Vultr cloud platform [21]. Machines are deployed automatically using a developed Python script that uses the Vultr API [22]. The machines have the following characteristics that correspond to the recommendations [23]:

– processor: 1 virtual processor;

– RAM: 8 GB;

– drive: 50 TB;

– operating system: Ubuntu 22.04 LTS.

5. Results of investigating the method of detection and identification of Rootstock network blockchain nodes

5. 1. Peer Discovery protocol analysis

Peer Discovery is the general name of protocols for finding nodes in P2P networks, which are often implemented using the distributed hash table Kademlia [24] on the UDP protocol. Rootstock network is no exception. The main logic of the protocol is implemented in the PeerExplorer class [25]. An example of the analysis is shown in Fig. 1.

The PeerExplorer class handles message processing and works with other parts of the protocol, such as a distributed table of nodes.

Fig. 1. PeerExplorer class analysis process

5. 1. 1. Distributed hash table

Kademlia DHT is implemented in the NodeDistanceTable class [26] and mixes slots with nodes according to their distance to the current node. Table parameters include [27]:

– 256 slots (buckets);

– 16 nodes in a slot.

An important element of the algorithm is the node identifier (*NodeId*). The node's RSK identifier is the node's public key without the first byte of the format [28]. The distance between the nodes is calculated based on this ID, or more precisely, based on the hash of the ID *keccak256*(*keccak256*(*NodeId*)). The distance is the position of the most significant bit of the value of XOR operation over the hashed values of the identifiers of two nodes (Fig. 2) [29].

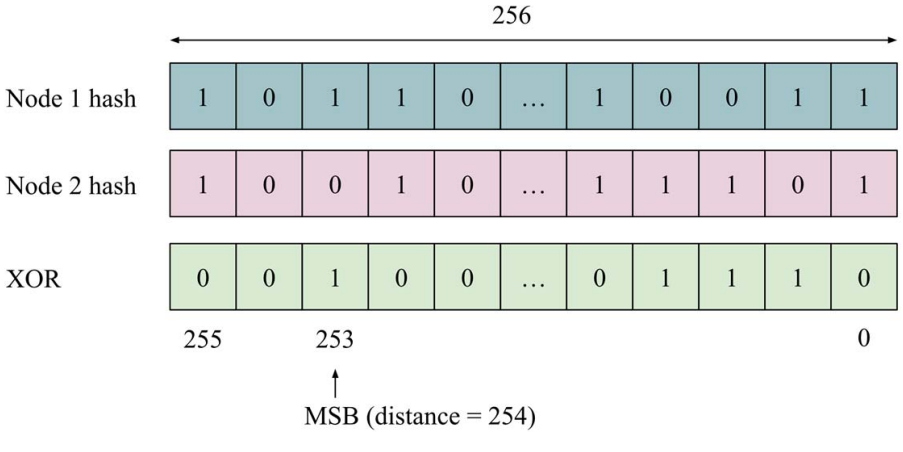

Fig. 2. Visualization of the calculation of distance between nodes

For a slot with a distance of 1, the hash of the node IDs is almost identical and differs only by one least significant (zero) bit. In practice, this means that more nodes will fall into slots with a longer distance than slots with a shorter distance.

5. 1. 2. Message format

The protocol is implemented using four types of RLP messages [30]: two types of requests and two types of responses. RLP messages are wrapped with additional information and sent. In general, the packet sent via the UDP protocol has the structure shown in Fig. 3.

MDC (32 bytes)	Signature (65 bytes)	Type (1 byte)	Data (N bytes)

Fig. 3. Peer Discovery message structure

The fields have the following values:

– Data – RLP message;

– Type – type of RLP message (PING, PONG, FIND_NODE or NEIGHBORS);

– Signature – ECDSA data signature (Type and Data fields), specified by parameters v (32 bytes), r (32 bytes), and s (1 byte). The signature uses the node's private key;

– MDC is a checksum that is formed on the basis of the rest of the data, namely, *keccak256*(*Signature+Type+Data*).

All RLP messages have 2 mandatory fields *rlpCheck* and *rlpNetworkId*. *rlpCheck* is a unique identifier (Universally Unique Identifier – UUID), used to check that the response came to the specified request, that is, the response must contain the same *rlpCheck* value as the request. *rlpNetworkId* is the network identifier; in the RSK network, possible values are 775 and 8100, for mainnet and testnet, respectively. The format of the 4 types of messages is given below:

– PING. Request response, used to check the availability of the node and to initialize the connection. Upon receiving this message, the receiving node sends a PONG response and also sends a PING if the connection between the nodes is not initialized.

Although a message of this type in other blockchain networks [19] involves specifying UDP (for this protocol) and TCP (for Wire protocol) ports separately, in *RSKj* the same port is used:

[[*rlpIp, rlpPort, rlpPort*], [*rlpIpTo, rlpPortTo, rlpPortTo*], *rlpCheck, rlpNetworkID*].

In addition, the message format is supposed to indicate the connection parameters of the sending node and the receiving node, RSKj specifies the parameters of the sending node instead of the receiving node, so the final message format takes the following form:

[[*rlpIp, rlpPort, rlpPort*], [*rlpIp, rlpPort, rlpPort*], *rlpCheck, rlpNetworkID*],

where *rlpIp* is the IP address of the sender of this message, *rlpPort* is the connection port;

– PONG. Reply to PING message. When receiving a PONG response, the receiving node tries to add the sending node to the Kademlia table. Adding to the table means that the connection between the nodes is established. If the bucket for the current distance is full, no addition to the table occurs and the connection is not considered established.

As in the PING message, the message duplicates the connection parameters of the message sender and uses one port for UDP and TCP protocols:

[[*rlpIp, rlpPort, rlpPort*], [*rlpIp, rlpPort, rlpPort*], *rlpCheck, rlpNetworkID*],

where *rlpIp* is the IP address of the sender of this message, *rlpPort* is the connection port;

– FIND_NODE. Query of neighboring nodes. The message has the following format: [*rlpNodeId, rlpCheck, rlpNetworkId*], where *rlpNodeId* is the ID of the node (based on the public key of the node) for which neighboring nodes will be provided;

– NEIGHBORS. Reply to FIND_NODE message. One message provides up to 20 neighboring nodes (*MAX_NODES_PER_MSG*) of Kademlia tables per message. Among them are 15 nearest neighbors [31] and 5 random ones [32]. Nearest neighbors are neighbors with the smallest logical distance between them and between *rlpNodeId* (specified by the request). The response is provided only if the node is considered connected, i.e., it is itself in the Kademlia distributed table.

The response has the following format: [[*node*1, *node*2, *node*2, ..., *node*20], *rlpCheck, rlpNetworkId*], where *node* is a neighboring node given by the following list [*rlpHost, rlpUDPPort, rlpTCPPort, rlpId*]. *rlpHost* – IP address or host name of the node, *rlpUDPPort* – UDP connection port, *rlpTCPPort* – TCP connection port, *rlpId* – node ID based on the node's public key.

5. 1. 3. Sequence of messages

A typical sequence for obtaining neighbors when node A queries node B for neighbors is as follows:

Step 1. Node A sends a PING request to Node B.

Step 2. Node B sends a PONG response to node A. Node A adds node B to its distributed table.

Step 3. Node B sends a PING request to node A.

Step 4. Node A sends a PONG response to Node B. Node B adds Node A to its distributed table. If the slot in the table for the distance between these nodes is full, no addition is made. Instead, the connection to the most recently pinged node is checked to remove it from the table if it is no longer active. At this stage, node A does not know whether node B has added it to the corresponding slot.

Step 5. Node A sends a FIND_NODE request to Node B.

Step 6. If node B contains node A in its connection table, then neighboring nodes are selected and sent in the NEIGHBORS message. If it does not contain it, then the response is not sent.

5. 2. The main stages of the node detection method

The main stages for searching for network nodes based on the breadth-first search algorithm are formed:

Step 1. Create a queue that will contain nodes from which you need to query neighboring nodes. Create a list of all received nodes.

Step 2. Specify the nodes from which to start the search. Add initial nodes to the queue and list of received nodes. The initial nodes are the same as those specified as boot nodes in the *RSKj* configuration file [33].

Step 3. Get a node from the queue from which to request nodes in the next steps.

Step 4. Connect to the node. To establish a connection, you need to send PING, receive PONG, receive PING, send PONG.

If the node to which the connection is made has all the slots in the distributed hash table for a given distance filled, then when processing the PONG response, it will not add a new node to the list of connections. Therefore, it will not respond to the new node's requests to provide neighbors.

This stage can be optimized: generate the private key of the node until the public key is obtained, which will make it possible to obtain a smaller distance *K* to the node to which the connection is made. Thanks to this, the node will add us to a closer slot, that is, a less crowded slot. This will increase the chances of establishing a connection on the first try;

Step 5. Ask neighboring nodes from the connected node. Send a FIND_NODE request and get a NEIGHBORS response. Add new neighbor nodes (that are not in the list of all received nodes) to the queue and to the list of received nodes.

Since the list of nodes transmitted in the response is non-deterministic (15 nearest neighbors and 5 random ones), it is impossible to statically determine the number of requests that need to be sent to get all the neighbors of the current node. Therefore, a heuristic approach is used to determine the number of messages: send messages until a new node is received for the last L messages, but at least M requests.

If the node does not respond to the node request, it may mean that the connection in Step 4 has not been established. In this case, return to the previous step (but a limited number of times);

Step 6. If the queue is not empty, return to Step 3. It is also possible to improve this step. Since the blockchain network is dynamic, nodes are constantly connected and disconnected from it. Therefore, it is possible to poll already processed nodes to re-obtain nodes with a certain periodicity.

5. 3. Development of a prototype of node detection software

The prototype of the node detection software is implemented in Python. To speed up the search, the algorithm is implemented multi-threaded using ThreadPoolExecutor and queue (synchronized queue). The coincurve library is used for asymmetric cryptography, namely, private key generation, message signature, and other operations. The pycryptodome library is used for hashing.

The software connects to multiple nodes simultaneously in 15 different threads and polls neighboring nodes. For better isolation, separate ports (from 1000 to 9999) are used for different nodes. Fig. 4 shows a schematic diagram where the software connects to multiple nodes simultaneously and uses different ports to receive messages.

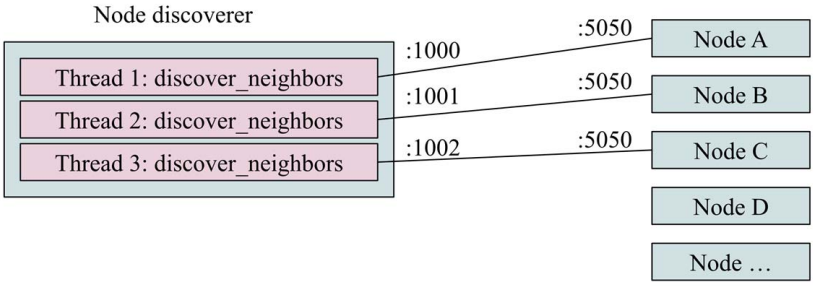

Fig. 4. Multi-threaded connection to nodes

The method software involves sending PING, FIND_NODE requests, and also sending a PONG response. The NEIGHBORS response is not sent, only received, and processed. In messages, the *rlpCheck* field is generated as a random *UUID*. Since the experiment is carried out in the main network (mainnet), the network identifier *rlpNetworkId* is set to 775.

When collecting nodes, the following information is registered:

– IP address of the node;

– UDP and TCP ports for Peer Discovery and Wire protocols;

– node identifier;

– node detection time.

The Geolocation DB web service API [34] is used to obtain IP address geolocation.

5. 4. Evaluating effectiveness of the proposed method

To check the effectiveness of the proposed method, the following experiment is conducted. The software of the developed method collects a list of all available blockchain nodes of the RSK network. To verify that it detects all nodes, test nodes are used, which are additionally deployed for the duration of the experiment with a certain period and in different physical locations. Discovery means receiving a node in the NEIGHBORS message.

Two variations of the developed method are used to search for nodes: a normal and an optimized version. The optimization consists in the fact that the public key that identifies us as a node is chosen in a special way. The private key is iterated randomly until a public key is generated that gives a distance to the node (whose neighbors are queried) that is less than or equal to 250 with a maximum of 256. This increases the chances of getting into an unfilled Kademlia table slot on the first try, since what the smaller the distance, the less filled the slot. For both variants of the method, FIND_NODE requests are sent to the same node until a new node is received in the last 100 messages, but not less than 1000 requests.

The results of the experiment are given in Table 1. The experiment uses 6 test nodes. Nodes are deployed within 3 hours. The node discovery software starts after all test nodes are deployed. During the experiment, the deployment time of the node and the time of its detection by the developed method are recorded.

Fig. 5 shows a plot of the detection of all unique network nodes over time. Optimizing the selection of the public key made it possible to significantly reduce the time of searching for nodes due to the absence of waiting for an available space in the table slot of neighboring nodes.

In general, the developed method revealed 220 (without optimization) and 222 (with optimization) unique nodes. Unique nodes are nodes that have a unique pair of IP address and node ID. That is, it is possible to have several nodes with different identifiers but one IP address. The detected nodes have 209 unique IP addresses. Fig. 6 shows their physical location using the geojson.io service [35].

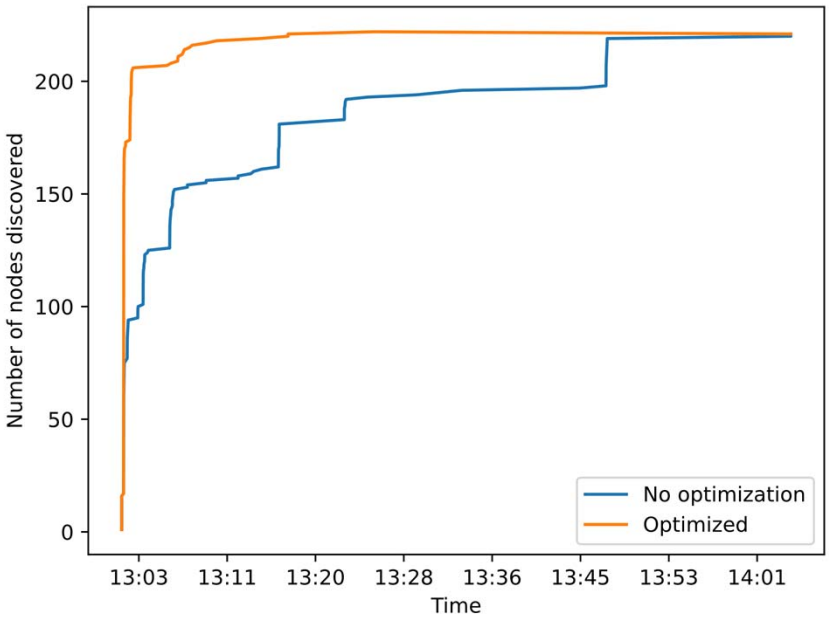

Fig. 5. Number of new nodes discovered over time

Table 1

Detection of test nodes

Node number	Physical location of the node (city)	Node deployment time	Software detection time	Detection time by optimized software
1	Sydney	10:09	13:03	13:02
2	Tokyo	10:48	13:01	13:01
3	Manchester	11:08	13:06	13:02
4	Seoul	12:00	13:03	13:02
5	Seattle	12:12	13:22	13:01
6	Warsaw	12:57	13:01	13:01

The collection of nodes took approximately 1 hour and the entire experiment with deployment of nodes took 4 hours.

Fig. 6. Physical location of node IP addresses

6. Discussion of results of investigating the method of detection and identification of nodes of the Rootstock blockchain network

Node discovery protocols are the foundation of any decentralized P2P network. They determine how nodes find other network nodes, ensure its distribution, and form a certain network topology. In such systems, nodes can connect to every node in the network or only to a subset of nodes. In blockchain systems, nodes usually do not store information about each node in the network and only connect to a small subset of nodes to optimize resources and communication efficiency.

P2P systems include file exchanges, communication systems, blockchain networks, and others. Depending on the functions of the system, a different network topology and the use of different node detection protocols are possible.

The protocol for detecting nodes in the Rootstock blockchain network is built on the basis of a table of neighboring nodes – Kademlia, which takes into account the logical distance between nodes. This distance is determined based on the hash values of the nodes' public keys. Visualization of the calculation of the distance between nodes is shown in Fig. 2. Kademlia is used in file sharing networks and various blockchain networks. It is suitable for decentralized systems that require a decentralized network topology. The advantages of Kademlia include the efficiency and speed of data retrieval, scalability, and low resource costs. Kademlia's disadvantages include security issues (such as vulnerability to Sybil attacks) and the difficulty of maintaining a dynamic network with constantly changing nodes.

However, there is no documentation of the Rootstock network node discovery protocol. Although the blockchain protocol of the Ethereum network is also used by Kademlia, it is not exactly identical to the node detection protocol of the Rootstock network. Therefore, the Ethereum protocol specification [20] or available studies [15, 16] cannot be directly used to build a crawler in the Rootstock network. That is why the analysis of the source code of the software of the RSKj node was carried out, which made it possible to gain an understanding of the structure and mechanisms of the protocol.

In the Rootstock network, the node discovery protocol is implemented using only 4 types of messages (2 for a request and 2 for a response). In addition, a specific sequence of messages was defined, which ensures detection and connection with neighboring nodes in the network. In general, the simplicity of the protocol and implementation is a strength of the network, as it reduces the probability of errors and simplifies the analysis process.

Since the nodes in the blockchain network are equal, and by default the parameters of the Peer Discovery protocol in *RSKj* are the same, it is impossible to determine the criteria by which a node can be considered a higher priority for providing neighboring nodes. Under this condition, if the network is represented as a graph, it will be considered unweighted.

To search for nodes, namely, traversal of an unweighted graph, it is possible to use both depth-first search and breadth-first search. Among the advantages of using breadth-first search is the ease of parallelization, where each thread simply fetches the next node from the queue. Therefore, a breadth-first search algorithm was chosen to search for blockchain network nodes.

An important element is that the size of the slots in the Kademlia table is quite small. Therefore, quite often situations may arise when a node cannot establish a connection with another node because the slot of the corresponding distance is full. In order for a node to have more chances to establish a connection with the selected node, the logical distance between them should be as small as possible. And since the distance is determined on the basis of node identifiers (public keys), the process of getting a node into a free slot was optimized by sorting keys to obtain a smaller distance. This optimization is absent in similar studies of the Ethereum network [7, 8, 11, 16, 17] and was implemented in this work as an additional element.

The next important aspect is the number of neighbor grant requests that need to be sent to a single node to obtain its neighbors. 15 nearest neighbors and 5 random ones are provided per message. Therefore, when sending any number of messages, it is impossible to be sure whether all neighboring nodes have received them or only some part. A heuristic approach is used for this: messages are sent until a new node (which has not yet been discovered by the given node) is received in the last 100 messages, but in any case, at least 1000 requests are sent. Since the blockchain network is dynamic, nodes are constantly connected and disconnected from it. Therefore, previously polled nodes are also polled to re-obtain nodes with a certain frequency.

When implementing the method, the multi-threaded ThreadPoolExecutor and queue mechanisms of the Python language were used to speed up the search. The software connects to multiple nodes simultaneously in 15 different streams, although the number of streams can be higher. It is even possible to allocate a separate flow for each node. However, in order to avoid potential blocking of requests by Intrusion Prevention Systems, it was decided to choose a limited number of threads – 15, which satisfy sufficiently fast collection of all nodes.

As a result of the experiment, information was collected about 222 nodes of the network, which includes IP addresses of nodes, UDP and TCP ports for Peer Discovery and Wire protocols, node IDs and node discovery time. In addition, verification nodes were deployed, the purpose of which was to check the effectiveness of the developed detection technique. All 6 test nodes were detected in less than 10 minutes. Data on the time of placement and detection of these test nodes are given in Table 1. The experiment also checked to what extent the selection of the public key of the node (for searching a shorter distance) optimizes the time of searching for nodes. Fig. 5 shows a graph of node detection time, which allows one to make sure that this optimization speeds up the search. With optimization, most nodes were detected within 15 minutes, while without optimization it took just over 45 minutes. Finally, the physical location of all detected nodes is shown in Fig. 6, which allows us to draw conclusions about the location of nodes. For example, the most nodes are located in South America, the United States, and Central Europe.

It is safe to say that the developed method turned out to be a highly effective tool for finding and identifying nodes in the Rootstock blockchain network. A special feature of this method is its ability to quickly detect a large number of nodes in the network, where it can find most of the active nodes in a matter of minutes. The developed node search method and the corresponding experiment made it possible to obtain information about the topology of the Rootstock network, which was not reported before.

The main limitation of the developed node detection method is its inability to identify real IP addresses of nodes when they use anonymizing services such as virtual private network (VPN) or The Onion Router (TOR). These services are designed to ensure the privacy and anonymity of users by hiding real IP addresses using various network routes and gateways. As a result, when a node uses VPN or TOR, the method can only identify the IP address of the source node of these services, not the actual IP address of the node. This significantly limits the capabilities of the method in the context of accurate tracking and analysis of network activity.

The development of this research may focus on the practical application of our method for specific tasks, for example, for the analysis of blockchain transactions or the network in general. Another direction of development may be the adaptation of the devised method to other P2P networks, especially those whose protocols function on the basis of mechanisms similar to those already studied. Another area of further research is the analysis of nodes that are in the VPN or TOR network in order to obtain any information about the real IP addresses of the nodes.

7. Conclusions

1. The Peer Discovery blockchain protocol of the Rootstock network was analyzed. A comprehensive approach was applied, which includes the analysis of the source code of the RSKj node software and an analytical review of the documentation of a conceptually similar protocol – the node detection protocol in the Ethereum network. This made it possible to understand how the Kademlia distributed node table is implemented and works, to obtain message types and formats, and to determine message sequences to obtain a list of neighboring nodes. In other words, the complete logic of the node search protocol has been restored. This information is sufficient for independent implementation of the protocol and communication in the Rootstock network blockchain with other nodes.

2. The main stages of the method of detecting nodes of the Rootstock network blockchain based on breadth-first search were defined. Nodes are interrogated by the Peer Discovery protocol to obtain neighboring nodes. Neighbor nodes are then polled to obtain their neighbor nodes. This procedure continues until new nodes are detected. Also, the search can be performed under continuous mode (nodes are polled repeatedly) to obtain information about the state of the network in real time. In addition to the basic algorithm, the optimization of the entry of a node into the slot of the shortest distance of the distributed table by selecting the public key (identifier) of the node has been developed. This allows one to speed up the search for nodes by increasing the chances that the specified slot in the table will be empty and the connection will be made on the first try.

3. The designed prototype of the Rootstock network node detection software implements the node search algorithm using multi-threaded mechanisms. This allows one to perform an accelerated search for nodes. When collecting nodes, the IP address of the node, its identifier (public key), the ports of Peer Discovery and Wire protocols, and the time of node discovery are obtained.

4. Effectiveness of the proposed method was verified using the developed software prototype and the following experiment in the main network. 6 verification nodes were deployed in different physical locations and at different times. The deployment period lasted up to 3 hours, after which a 1-hour node search was initiated. All 6 test nodes were detected in less than 10 minutes. During the entire search time, the developed method revealed 220 (without optimization) and 222 (with optimization) unique nodes. Optimizing the selection of the public key made it possible to reduce the time of searching for nodes by almost 3 times due to the absence of waiting for an available space in the table slot of neighboring nodes. In addition, with the help of existing services, the physical location of all 209 unique IP addresses of the nodes was determined and a corresponding map was built.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

The manuscript has associated data in the data warehouse.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

References

1. Dorogiy, Y., Kolisnichenko, V. (2023). Application of logging in various participants of blockchain networks for de-anonymization of the end user. Herald of Khmelnytskyi national university, 1 (5), 60–66. Available at: http://journals.khnu.km.ua/vestnik/?p=20028
2. Ethereum staking. Available at: https://ethereum.org/staking
3. Smart contracts secured by Bitcoin. Available at: https://rootstock.io/
4. Lerner, S. D. (2020). Building the Most Secure, Permissionless and Uncensorable Bitcoin Peg. Available at: https://medium.com/iovlabs-innovation-stories/building-the-most-secure-permissionless-and-uncensorable-bitcoin-peg-b5dc7020e5ec
5. Howell, A., Saber, T., Bendechache, M. (2023). Measuring node decentralisation in blockchain peer to peer networks. Blockchain: Research and Applications, 4 (1), 100109. https://doi.org/10.1016/j.bcra.2022.100109
6. Grundmann, M., Amberg, H., Hartenstein, H. (2021). On the Estimation of the Number of Unreachable Peers in the Bitcoin P2P Network by Observation of Peer Announcements. arXiv. Available at: https://doi.org/10.48550/ARXIV.2102.12774
7. Eisenbarth, J.-P., Cholez, T., Perrin, O. (2022). Ethereum's Peer-to-Peer Network Monitoring and Sybil Attack Prevention. Journal of Network and Systems Management, 30 (4). https://doi.org/10.1007/s10922-022-09676-2
8. Henningsen, S., Teunis, D., Florian, M., Scheuermann, B. (2019). Eclipsing Ethereum Peers with False Friends. arXiv. Available at: https://doi.org/10.48550/ARXIV.1908.10141
9. Deshpande, V., Badis, H., George, L. (2018). BTCmap: Mapping Bitcoin Peer-to-Peer Network Topology. 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN). https://doi.org/10.23919/pemwn.2018.8548904
10. Xu, D., Gao, J., Zhu, L., Gao, F., Zhao, J. (2023). Statistical and clustering analysis of attributes of Bitcoin backbone nodes. PLOS ONE, 18 (11), e0292841. https://doi.org/10.1371/journal.pone.0292841
11. Li, K., Tang, Y., Chen, J., Wang, Y., Liu, X. (2021). TopoShot: Uncovering Ethereum's Network Topology Leveraging Replacement Transactions. arXiv. Available at: https://doi.org/10.48550/ARXIV.2109.14794
12. Miller, A. K., Litton, J., Pachulski, A., Gupta, N., Levin, D., Spring, N., Bhattacharjee, B. (2015). Discovering Bitcoin's Public Topology and Influential Nodes. Available at: https://www.cs.umd.edu/projects/coinscope/coinscope.pdf
13. Rohrer, E., Tschorsch, F. (2019). Kadcast. Proceedings of the 1st ACM Conference on Advances in Financial Technologies. https://doi.org/10.1145/3318041.3355469
14. Neudecker, T. (2019). Characterization of the Bitcoin Peer-to-Peer Network (2015-2018). Karlsruhe. Available at: https://doi.org/10.5445/IR/1000091933
15. Eyong, M. (2019). Analyzing the Peer-to-Peer Network of Ethereum. Available at: https://www.researchgate.net/publication/354149867_Analyzing_the_Peer-to-Peer_Network_of_Ethereum
16. Wang, T., Zhao, C., Yang, Q., Zhang, S., Liew, S. C. (2021). Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain. IEEE Transactions on Network Science and Engineering, 8 (3), 2131–2146. https://doi.org/10.1109/tnse.2021.3078181
17. Wolchok, S., Halderman, J. (2010). Crawling BitTorrent DHTs for fun and profit. 4th USENIX Workshop on Offensive Technologies (WOOT '10), Washington, D.C. Available at: https://jhalderm.com/pub/papers/dht-woot10.pdf
18. rskj: RSKj is a Java implementation of the RSK protocol. Available at: https://github.com/rsksmart/rskj
19. Node Discovery Protocol. Available at: https://github.com/ethereum/devp2p/blob/master/discv4.md
20. Sedgewick, R., Wayne, K. (2011). Algorithms. Addison-Wesley Professional.
21. SSD VPS Servers, Cloud Servers and Cloud Hosting. Available at: https://www.vultr.com/
22. Vultr API. Available at: https://www.vultr.com/api/
23. Hardware requirements. Available at: https://dev.rootstock.io/rsk/node/install/requirements/
24. Maymounkov, P., Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. Lecture Notes in Computer Science, 53–65. https://doi.org/10.1007/3-540-45748-8_5
25. PeerExplorer.java. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/java/co/rsk/net/discovery/PeerExplorer.java
26. NodeDistanceTable.java. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/java/co/rsk/net/discovery/table/NodeDistanceTable.java

27. KademliaOptions.java. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/java/co/rsk/net/discovery/table/KademliaOptions.java
28. ECKey.java. Available at: https://github.com/rsksmart/rskj/blob/6dde0cdeeb2138e61dc845810eaa8ce55a8d2b7f/rskj-core/src/main/java/org/ethereum/crypto/ECKey.java#L287
29. DistanceCalculator.java. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/java/co/rsk/net/discovery/table/DistanceCalculator.java
30. PeerDiscoveryMessageFactory.java. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/java/co/rsk/net/discovery/message/PeerDiscoveryMessageFactory.java
31. PeerExplorer.java. L221. Available at: https://github.com/rsksmart/rskj/blob/e06686fe83554c6381db207857e13b6e76e79ace/rskj-core/src/main/java/co/rsk/net/discovery/PeerExplorer.java#L221
32. PeerExplorer.java. L319. Available at: https://github.com/rsksmart/rskj/blob/e06686fe83554c6381db207857e13b6e76e79ace/rskj-core/src/main/java/co/rsk/net/discovery/PeerExplorer.java#L319
33. main.conf. Available at: https://github.com/rsksmart/rskj/blob/master/rskj-core/src/main/resources/config/main.conf#L32
34. Geolocation DB. Available at: https://geolocation-db.com/json/
35. geojson.io. Available at: https://geojson.io/