Interaction support system of network aplications

Oleksii R. Rudkovskyi[a], Galina G. Kirichek[a,b]

[a]Donetsk National Technical University, 2 Shybankova Sq., Pokrovsk, 85300, Ukraine
[b]National University “Zaporizhzhia Polytechnic”, 64 Zhukovsky St., Zaporizhia, 69063, Ukraine

Abstract
In this article implemented a method of organizing a distributed network to launch and support the work of applications. In the process of building the network model, different algorithms were used, at the same time encryption algorithms are used (Elliptic curve Diffie–Hellman and Advanced Encryption Standard), Secure Hash Algorithm 1 algorithms for hashing, onion routing algorithm and other. Transmission Control Protocol transport protocol is used to send data between nodes of the network. Methods and application for creating and launching other applications have been implemented in the Java language. Linux containers and Docker are used to isolate different applications running on the network. In this way, different applications written using different programming languages can run without affecting other running applications and client’s operation system. Application’s data is stored on the network distributed with and without encryption.

Keywords
Docker, distributed networks, encryption, Java, TCP

1. Introduction

In today’s world it is impossible to imagine the operation of data transmission devices for various purposes without a network connection. It is even more difficult to imagine performing a large number of tasks without using the usual services, such as e-mail, cloud storage, various messengers, web applications, etc. In addition, the development of data transmission systems has reached a level where almost any object can be connected to the network. This constantly increases the amount of traffic and data that needs to be stored or processed. Therefore, with the development of information technology and data transmission methods, the issues of further development of networks and support of secure data transmission systems are becoming more and more urgent and acute [1].

One of the problems of the modern Internet is the rapid development of the Internet of Things. According to the latest forecasts for the next few years, the number of devices connected to the Internet will increase several times. With the new version of the internet protocol (IP) IPv6, the number of addresses compared to IPv4, increased at times, but in the near future, this volume of addresses may be exhausted, according to the rate of growth of the Internet

CS&SE@SW 2020: 3rd Workshop for Young Scientists in Computer Science & Software Engineering, November 27, 2020, Kryvyi Rih, Ukraine
alexteen12@gmail.com (O.R. Rudkovskyi); kirgal08@gmail.com (G.G. Kirichek)
https://orcid.org/0000-0002-5267-2525 (O.R. Rudkovskyi); http://csn.zntu.edu.ua/galina-grigorivna-kirichek (G.G. Kirichek)
0000-0002-5267-2525 (O.R. Rudkovskyi); 0000-0002-0405-7122 (G.G. Kirichek)

of things. The reasons for the acceleration of filling IPv6 also include legislation of individual countries or alliances, the development of astronautics and the colonization of other planets [2].

Currently, almost all applications run in the form of client-server services. These services allow you to exchange messages, transfer files, receive news and publish them. Services can interact with both users and other services. But now the services are poorly protected and, after attacks by attackers, may lose confidential information left by users. In addition to public information, attackers steal bank card data, private files, etc., so the protection of this data is a priority for companies. In addition, it is constantly necessary to address the availability and speed of information transfer [3].

Thus, it is relevant to solve the problems of increasing the speed of information processing, supporting the security of transmission, storing and processing data, as well as executing server applications as close as possible to the client in order to reduce the delays between executing requests and receiving responses (critical for the operation of some categories of IoT devices).

2. Objectives and solutions

2.1. Objectives

The purpose of this work is to research and implement software that can support communication of network devices and applications, by developing protocol of distributed system to improve work on services and their communication with devices. The object of this research is the process of software implementation that support distributed communication between devices and applications. The subject of research are the models, methods and software tools for organizing the interaction of devices and applications in a distributed environment.

Currently, familiar virtual or dedicated machines (VPS and VDS) are used as a server infrastructure for applications, but recently containers are gaining more and more popularity due to the ease of scaling, security and speed of work [4, 5]. At the same time, the Internet is actively developing decentralized and distributed system on which it is also possible to create applications. These applications use a distributed database (usually blockchain), which guarantees the security of data storage, and allow interaction with applications with different devices and programs that are capable of performing certain calculations to participate in the system itself. The problem is that traditional technologies are not secure enough, and new distributed systems do not provide the same capabilities for creating and interacting with programs that traditional ones provide. For example, the Ethereum network provides the ability to develop applications that are able to receive data from outside and trigger events that other programs can process, but these programs are extremely limited in functionality and are not able to interact with other devices on the Ethereum network or with devices from Internet and are not capable of performing any resource-intensive tasks [6, 7].

The research is aimed at finding a solution that combines two approaches to developing and launching applications, providing a high level of security in the form of a distributed network and providing many opportunities for developers as traditional servers provides.

As the platform of distributed network many different independent devices (personal computers, mobile phones, servers, etc.), which contain launched specific software, is used. On launching the application and while it works, devices connecting to each other and always ready to participate in processing or transferring a data. Device can support multiple roles: be a binder (intermediate) node, store data, launch applications or be a client who requests some calculations. Special algorithms are used to define a role for each node.

Before launching client's application, network should detect involved, active devices, find needed for processing requests nodes, connect involved nodes into one network and, after that, should launch needed applications. At the same time, while working, devices additionally divided to next types: computer node (executes needed application, in network can be one or more nodes of this type); client node (node, which requests calculations or that can receive calculated data); router (intermediate device that connect other nodes and control process of calculation).

For different types of nodes authors recommends to use different types of packets. In this case, router (intermediate device) should support safety of data transmission without having access to them in the same time. Based on the fact, that the network is distributed, we consider in advance any node in the network to be compromised and unsafe. Therefore, considering this fact, for solving safety problem when transferring data in distributed network in work proposed to use special algorithms for transferring and encrypting data.

2.2. Solutions for building a system

Consider several popular programming languages and applications that are suitable for realization of this task.

Java - a cross-platform programming language used for desktop and mobile devices supports a set of existing libraries for cryptography and network. Python is also a cross-platform, interpreted language, but slower. C++ is a low-level language, productive and with a large number of ready-made libraries, but more difficult to write program code [8]. After the analysis, the Java programming language was selected as cross-platform, object-oriented, popular, simple and fast enough [9, 10]. It allows you to run an application on many operating systems (Windows, Linux, macOS, etc.).

In the analysis of existing methods and encryption algorithms are considered the most popular AES, RSA, 3DES, and ECDH. A high level of security is provided by the asynchronous RSA or ECDH algorithm. At the same time, ECDH has high security with a shorter key length, and the RSA algorithm works faster. Since both algorithms do not allow encrypting large amounts of data, to solve the problem, we use a common encryption key and the synchronous AES algorithm as the safest and fastest [11, 12].

When choosing an isolated environment, virtualization methods using virtual machines and Linux containers are considered [2]. Virtualization consumes a lot of resources, which is why the number of concurrent virtual areas is very limited. Containers give access to all the resources of a node, are faster, and support the simultaneous operation of dozens of isolated areas [13]. When conducting research, Docker was chosen as an isolated environment, using Linux containers, as the most productive and not limiting the work of programs [13].

TCP is used as a data transfer protocol. For testing the system, a virtual machine created in VirtualBox with Ubuntu installed was selected. As a platform for transferring data overlay network is used. In this case, data transmission functions are assigned to the lower levels, thus abstracted from the network, operating systems and technologies. The same data is sent over the same algorithm regardless of the kind of network technology used by the client.

The next functionality that should be implemented in the network is the self-organization of nodes. As nodes are independent of each other, network should automatically detect role of each node. For this purpose we use indicators of a node: free disk space (taking into account the limitations of each individual client of the network), installed software, number of processors and core, RAM size.

If storage is not full, by more than a certain percentage, node can receive files for storage, what he notify the nearest devices about. Otherwise, node work as distributor - he can send files on request, if they exists in local storage, but can't receive new files. In case if node should receive new file, but storage is full, node makes a decision to clear storage and delete unnecessary information. To do that, node asks the network for deleting a file. Only after receiving confirmation from some number of devices that file can be deleted, node deletes this file [14].

As different operating system and end nodes can launch different set of programs, a node store information about software that he can launch. Program, which should be executed, wherein store minimal set of needed and observed parameters. While launching a program, node make verification and, in case if his parameters are satisfy minimal requirements, a program starts. Otherwise, the request is redirected to the network until a node is found that can launch the application.

On a device at the same time can work many client's applications with different roles. Each application has a unique address within the network. With the constant growth of the number of devices connected to the global network, the use of existing routing methods, taking into account the peculiarities of the network being implemented, is impossible. Although IPv6 can satisfy a large number of users on the Internet, in the future, the process of combining and simultaneously using different networks (Ethernet and Bluetooth) may require an internal routing system. In this case, it is proposed to use the public key of the encryption algorithm as the node address in the implemented network. This approach allows both to securely encrypt data, since the encryption keys are not sent in the usual form, and to provide multiple addresses for one physical device. In addition, using this approach guarantees anonymity, since it is impossible to determine the user's location address, or any other information about the user.

Therefore, to generate a shared key, while device working in the network and encrypt a data, we use a pair keys (public and private) and public key of another client. Since the public key is an address of a device, and pair keys are stored only on one device and never passed on to anyone, this approach is most secure. After generating a shared key, the application encrypts the data sent with it. Second client, after generating a key using his own pair keys and address of the device who sent a data, will receive the same key [15].

In this case, if the sender and the recipient of the data are specified correctly, that is, the data is not sent from a foreign node, which is masked as the sender, the recipient can decrypt the data without any problems. This also applies to protect data from substitution or alteration, since in case of data change you also need to have access to the sender's key pair to generate the correct shared key and to encrypt the message itself. Moreover, if the data is distorted, the recipient will not be able to decrypt it, or will decrypt it with interference. Thus, the process of encryption and decryption makes the system itself sufficiently protected from interception

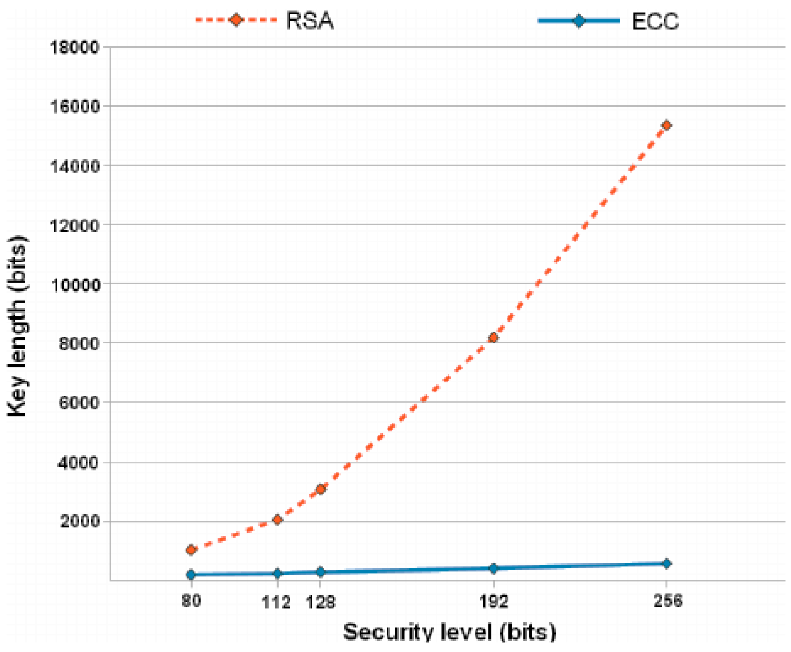


Figure 1: Comparison of RSA and ECDH algorithms

and data substitution.

For key pair and shared key generation cryptographic algorithm ECDH is used [16]. As you can see in figure 1, if compare to other algorithms, for example with RSA, the ECDH algorithm has a higher level of security with the same key size.

Although the ECDH algorithm is safe, it nevertheless does not allow encrypting large amounts of data since the maximum size of input data is limited by the key length. In addition, the creation of an encrypted message takes a long time, compared to other algorithms. To solve this problem, AES encryption algorithm and a shared key for the two devices are usually used. Algorithm AES can be considered cryptographically secure, and shared key generation without sharing parts of the keys through network makes it much secure and faster, depriving it of its main drawback - transmission of the key over the network [17, 18].

3. Stages of System Implementation

The nodes themselves can act as routers for devices that are unable to generate keys and encrypt data quickly enough. For example, microcontrollers cannot do all needed calculations fast enough, so as not to slow down its own functions.

3.1. Process of generation key pair for devices

To solve this issue node, to which a similar device is connected to directly, generates new keys that are associated with this device. Thus, devices get their own keys and become indistinguishable for other clients. Algorithm of generating these keys illustrated in figure 2.

Using these algorithms and methods allows using of the implemented network with almost any end device, regardless of its own performance [19]. The used methods make it possible to safely transfer data in a given network, while the computation of data, at almost any end node of the system, minimizes the delay in data transmission between the client and the node that processes the data itself, since such a node can be a neighboring device located in close proximity to the client.

In addition, these methods and algorithms guarantee the anonymity of clients and adapted to work in the notoriously compromised environment.

3.2. Description of client software modeling process

During the research and before implementation, the client software was modeled and a diagram of the required classes was obtained, which is shown in figure 3.

For proper operation of the client running on it and perform all functions of the system application needs to support needed interfaces regarding to operating system and end device, that runs the application.

Since the system provides information transfer network, it is necessary to describe the format of information.

3.3. The process of launching applications

The main tasks of the application are the exchange commands and their processing. The command is transmitted using the JSON data format, which is cross-platform and allows the same identification of data regardless of the platform and programming language.

The command consists of four parts: the length of the header, header, data and an additional field. The header length field stores information about the command transmission path, the size of the

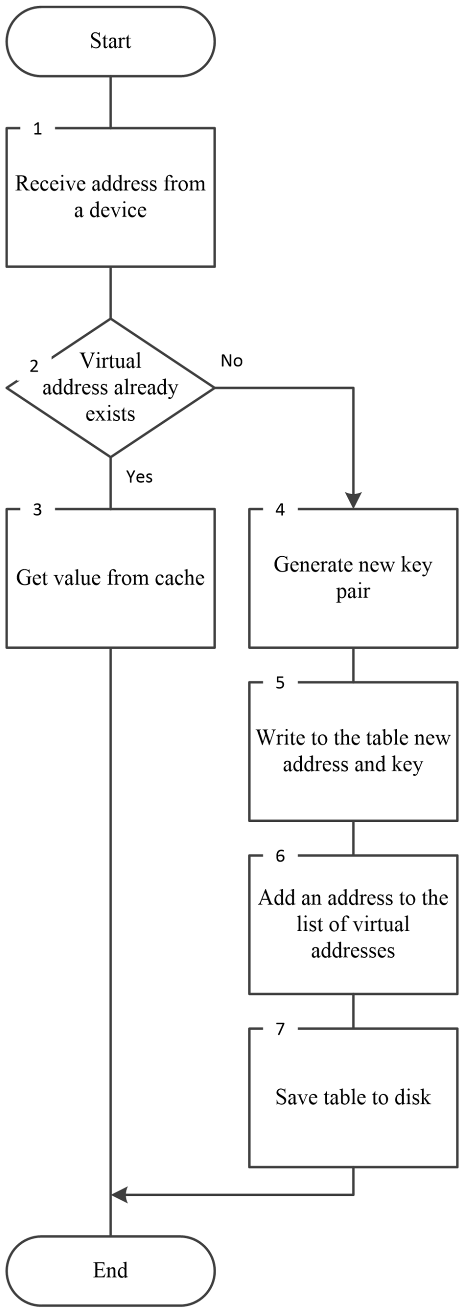
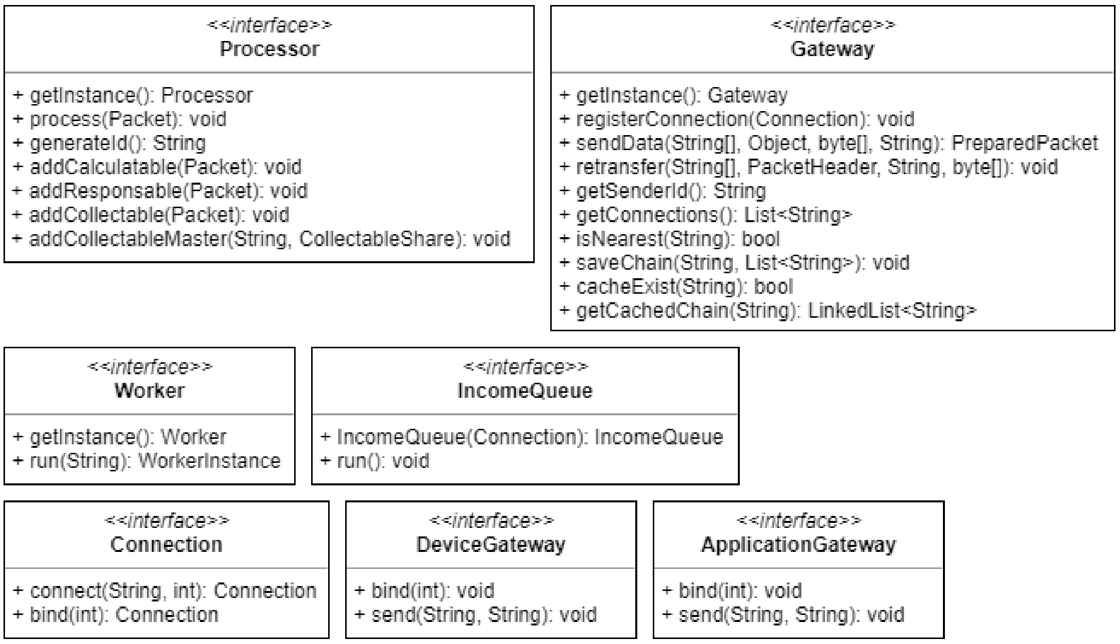

Start
1
Receive address from a device
2
Virtual address already exists
No
Yes
3
Get value from cache
4
Generate new key pair
5
Write to the table new address and key
6
Add an address to the list of virtual addresses
7
Save table to disk
End

Figure 2: Process of generation key pair for devices



<<interface>>
Processor
+ getInstance(): Processor
+ process(Packet): void
+ generateId(): String
+ addCalculatable(Packet): void
+ addResponsable(Packet): void
+ addCollectable(Packet): void
+ addCollectableMaster(String, CollectableShare): void
<<interface>>
Gateway
+ getInstance(): Gateway
+ registerConnection(Connection): void
+ sendData(String[], Object, byte[], String): PreparedPacket
+ retransfer(String[], PacketHeader, String, byte[]): void
+ getSenderId(): String
+ getConnections(): List<String>
+ isNearest(String): bool
+ saveChain(String, List<String>): void
+ cacheExist(String): bool
+ getCachedChain(String): LinkedList<String>
<<interface>>
Worker
+ getInstance(): Worker
+ run(String): WorkerInstance
<<interface>>
IncomeQueue
+ IncomeQueue(Connection): IncomeQueue
+ run(): void
<<interface>>
Connection
+ connect(String, int): Connection
+ bind(int): Connection
<<interface>>
DeviceGateway
+ bind(int): void
+ send(String, String): void
<<interface>>
ApplicationGateway
+ bind(int): void
+ send(String, String): void

Figure 3: System class diagram

data field and the additional field in bytes, as well
as information necessary for the correct processing of the request and is used for the correct delivery of the header itself. The data field stores the payload as a command if it is transmitted. Only the header and data are sent in JSON format, in plain text. The additional field stores data that cannot be expressed as text (such as binaries). Having received and correctly read all four parts of the command, the application can process it correctly.

If the command is for the current device, the text from JSON is converted to an object of the desired type. Further, depending on the object, either a specific class method is called, or the data is passed on [20, 21].

To launch additional client's applications, a special system command is used. It specifies application and status IDs, as well as initial data. Upon receiving the command, the node reads from the system the application description file, which contains the fields: tag image of the container; command to run the application; link to the first part of the application file and the hash sum of the file (optional); a list of auxiliary files and links to the initial bytes of these files and the hash sum of the file (optional). Upon receiving the file, the node checks whether the image has already been downloaded. If the image is loaded earlier – goes to the next step, otherwise - reads the image and in case of error interrupts the work and reports the error to the client. The following is the method of checking and loading the image:

```
public boolean isImageStoredLocally(String tag) {
    Process process = null;
    System.out.println("Checking tag: " + tag);
    try {
        process = Runtime.getRuntime().exec("docker images -q
        " + tag);
    } catch (IOException e) {
        return false;
    }
    BufferedReader bufferedReader = new BufferedReader(new
    InputStreamReader(process.getInputStream()));
    try {
        String imageId = bufferedReader.readLine();
        if (imageId == null) {
            return false;
        }
        return true;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
        }
    }

public boolean pullImage(String tag) {
    String command = " docker pull " + tag + " > /dev/null
    2>&1 && echo \"success\" || echo \"failed\"";
    Process process = null;
    System.out.println("Checking tag: " + tag);
    try {
        process = Runtime.getRuntime().exec("docker images -q
        " + tag);
    } catch (IOException e) {
        System.out.println("Can't execute command: " + command);
        return false;
    }
    BufferedReader bufferedReader = new BufferedReader(new
    InputStreamReader(process.getInputStream()));
    try {
        String response = bufferedReader.readLine();
        return response.equals("success");
    } catch (IOException e)
        {
        e.printStackTrace();
        }
        return false;
    }
```

After downloading the image, the node generates a unique identifier – the startup ID. This

ID is used to interact with the application. In order to send application data, you need to know both the address of the node on which the application is running and the startup ID, because multiple copies of the application can run on different nodes for different users.

To establish a connection between an application running in isolation and an application that manages a network client, the node listens for standard output from the isolated environment. If the first line contains the required sequence, then a connection with the application is established, since it is assumed that at this time the applications are already ready to connect to each other [22, 23]. All other lines are redirected to the user who requested the execution. However, the application does not write anything to the sandbox standard input, which prevents applications from running in terminal mode. The following is a simplified source code of the method for launching an application in an isolated environment and connecting to the application that is executed:

```
if (isImageExists) {
   try {
       Process dockerProcess = docker.run(appDescriptor.container,
       appInstanceId, appDescriptor.command, this.args);
       System.out.println("==> Connecting to application");
       BufferedReader bufferedReader = new BufferedReader(new
       InputStreamReader(dockerProcess.getInputStream()));
       String line = null;
       boolean isFirstLine = true;
       while ((line = bufferedReader.readLine()) != null) {
           if (isFirstLine) {
               appSocket = new AppSocket(this);
               if (appSocket.isConnected())
               {
                appSocket.setOnCommand(this::onCommandFromContainer);
                   appSocket.bind();
               }
               String[] args = {};
               appSocket.sendCommand("onStart", args);

        ApplicationEvent applicationEvent = new ApplicationEvent();
        applicationEvent.eventArgs = args;
        applicationEvent.eventName = "onStart";
        Gateway.getInstance()
                 .sendData(packetHeader.reverse,
                 applicationEvent, new byte[0],
                 packetHeader.targetId)
                     .send();
        isFirstLine = false;
           }
           else
```

```
            {
              response.line = line;
          Gateway.getInstance()
                  .sendData(packetHeader.reverse, response,
                  new byte[0], packetHeader.targetId)
                         .send();
            }
        }
        System.out.println("==> App finished his work");
        docker.cleanUp(appInstanceId);
    }
         catch (IOException e)
         {
        e.printStackTrace();
         }
    }
```

After starting the sandbox, applications have access to almost all the capabilities of the end device on which they run. For example, an application has access to multithreading, RAM and memory, video memory and video adapter, to the network itself and devices connected to it. However, at the same time, the application does not have access to the rest of the operating system, i.e. it cannot receive anything outside its area. It can see the total amount of used memory, but it cannot see what exactly the memory is occupied. In this case, it is impossible to access the user's files.

It is also impossible to get information on connections outside the isolated environment, but it is possible within the isolated part. In fact, the application works in a virtual area, but without a virtualization layer and therefore does not have performance limitations in comparison with conventional virtual machines, but, if necessary, can use any capabilities of the operating system. In addition, it is possible to install third-party software, which is subsequently removed along with the removal of the sandbox and all data that is generated by the application and which is not stored in distributed memory or another area.

3.4. Applying and testing the system

Research was carried out on a Mac mini. ESP8266 and ESP32 boards were used as devices, which were connected to a computer using a WiFi network. For testing, a program developed that calculates the factorial of large numbers (figure 4).

The developed system can be applied wherever there may be any calculations. Using this system, low-power devices can request heavy computations, which significantly expands the list of tasks that these devices can solve.

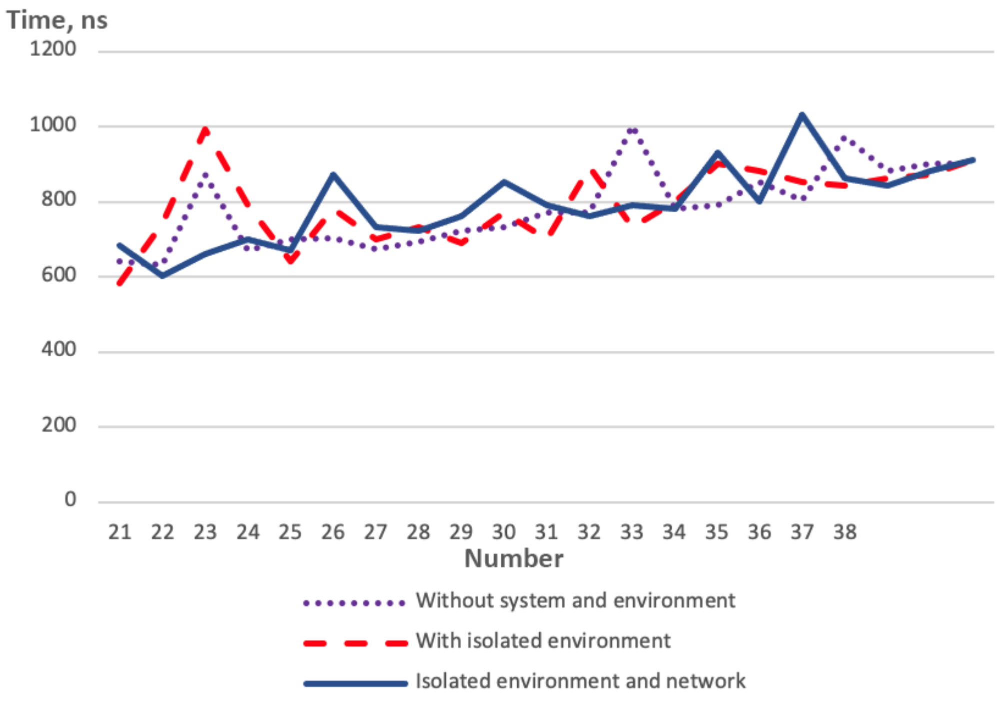


Figure 4: Comparing calculation speed

4. Conclusions

In this article improved the devices and applications interaction process in distributed systems, a model of a distributed system is created, and implemented a software protocol to support the interaction of network devices and applications. Methods of interaction of system parts and application components are developed.

The security of data transmission and their processing are one of the key in modern systems and data transmission networks, therefore, the greatest attention is paid to solving this problem in the process of implementing the system and algorithms of its operation. Application of the developed encryption and data transmission methods are increasing security of data transmission due to multi-layer encryption, and the use of an isolated environment makes the process of executing client applications independent of the host operating system and other applications that can be launched and can interfere with or intercept data. In the process of further research, it is planned to expand the capabilities of the system, namely to add support for graphical applications, parallelize processes on network devices and implement more functionality for interacting with the system and IoT devices.

References

[1] A. Amer, N. Abu, Comparison study between ipv4 & ipv6, International Journal of Computer Science Issues (IJCSI) 9 (2012) 314–317.

[2] G. Kirichek, V. Tymoshenko, O. Rudkovskyi, S. Hrushko, Decentralized system for run services, in: Proceedings of the Second International Workshop on Computer Modeling and Intelligent Systems, CMIS '2019, CEUR Workshop Proceedings 2353, Zaporizhzhia, Ukraine, 2019, pp. 860–872.

[3] G. Kirichek, S. Skrupsky, M. Tiahunova, A. Timenko, Implementation of web system optimization method, in: Proceedings of the Third International Workshop on Computer Modeling and Intelligent Systems, CMIS '2020, CEUR Workshop Proceedings 2608, Zaporizhzhia, Ukraine, 2020, pp. 199–210.

[4] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: 2015 IEEE international symposium on performance analysis of systems and software, ISPASS '2015, IEEE, Philadelphia, PA, USA, 2015, pp. 171–172. doi:10.1109/ISPASS.2015.7095802.

[5] T. Mauro, Adopting Microservices at Netflix: Lessons for Architectural Design, 2015. https://www.https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

[6] A. Gencer, S. Basu, I. Eyal, R. V. Renesse, E. Sirer, Decentralization in bitcoin and ethereum networks, in: International Conference on Financial Cryptography and Data Security, FC '2018, Springer, Berlin, Heidelberg, 2018, pp. 439–457. doi:10.1007/978-3-662-58387-6_24.

[7] A. Antonopoulos, G. Wood, Mastering Ethereum: Building Smart Contracts and DApps, O'Reilly Media, Inc, 2018.

[8] G. Kirichek, V. Kurai, Implementation quadtree method for comparison of images, in: 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering, TCSET '2018, IEEE, Slavske, Ukraine, 2018, pp. 129–132. doi:10.1109/TCSET.2018.8336171.

[9] M. Kumar, Energy Efficiency of Java Programming Language, Wayne State University, 2018.

[10] D. Abts, Masterkurs Client/Server-Programmierung mit Java: Anwendungen entwickeln mit Standard-Technologien, Springer-Verlag, 2019.

[11] C. P. Mayer, Security and privacy challenges in the internet of things, Electronic Communications of the EASST 17 (2009) 1–12. doi:10.14279/tuj.eceasst.17.208.

[12] W. Jansen, T. Grance, Guidelines on security and privacy in public cloud computing, in: NIST Special Publication 800-144, U.S. Department of Commerce, NIST Pubs, 2011.

[13] N. Pathania, Setting up jenkins on docker and cloud, in: Pro Continuous Delivery, Apress, Berkeley, CA, 2017, pp. 115–143.

[14] M. Miraz, M. Ali, P. Excell, R. Picking, Internet of nano-things, things and everything: future growth trends, in: Future Internet, volume 10, 2018.

[15] G. Kirichek, D. Kyrychek, S. Hrushko, A. Timenko, Implementation the protection method of data transmission in network, in: IEEE International Conference on Advanced Trends in Information Theory, ATIT '2019, IEEE, Kyiv, Ukraine, 2019, pp. 129–132. doi:10.1109/

ATIT49449.2019.9030482.

[16] A. Corbellini, Elliptic curve cryptography: a gentle introduction, 2015.

[17] B. Gulmezogluk, M. Inci, G. Irazoqui, T. Eisenbarth, B. Sunar, Cross-vm cache attacks on aes, in: IEEE Transactions on Multi-Scale Computing Systems, volume 10, IEEE, 2016, pp. 211–222. doi:10.1109/TMSCS.2016.2550438.

[18] S. Kumark, M. Girimondo, A. Weimerskirch, C. Paar, A. Patel, A. Wander, Embedded end-to-end wireless security with ecdh key exchange, in: IEEE 2003 46th Midwest Symposium on Circuits and Systems, volume 2, IEEE, Cairo, Egypt, 2003, pp. 786–789. doi:10.1109/MWSCAS.2003.1562404.

[19] G. Kirichek, V. Harkusha, A. Timenko, N. Kulykovska, System for detecting network anomalies using a hybrid of an uncontrolled and controlled neural network, in: Proceedings of the Computer Science & Software Engineering: Proceedings of the 2nd Student Workshop, CS&SE@SW '2019, CEUR Workshop Proceedings 2546, Kryvyi Rih, Ukraine, 2019, pp. 138–148.

[20] S. Sriparasa, JavaScript and JSON essentials, Packt Publishing, 2013.

[21] F. Pezoa, J. Reutter, F. Suarez, M. Ugarte, D. Vrgoč, Foundations of json schema, in: Proceedings of the 25th International Conference on World Wide Web, WWW '16, 2016, pp. 263–273. doi:10.1145/2872427.2883029.

[22] D. Merkel, Docker: lightweight linux containers for consistent development and deploymentno-things, things and everything: future growth trends, in: Linux journal, volume 2014, 2014, pp. 1–5.

[23] T. Bui, Analysis of docker security, in: Aalto University T-110.5291 Seminar on Network Security, arXiv preprint arXiv: 1501.02967v1, 2015.