Decentralized system for run services

Galina Kirichek[1[0000-0002-0405-7122]], Volodymyr Tymoshenko[1[0000-0001-8161-5119]], Oleksii Rudkovskyi[1[0000-0002-5267-2525]], Svitlana Hrushko[1[0000-0002-0064-408X]],

[1]Zaporizhzhia National Technical University, Zhukovsky str., 64, Zaporizhzhia, 69063, Ukraine
kirgal08@gmail.com,PigerDeus@gmail.com,alexteen12@gmail.com, grushko_ss@i.ua

Abstract. In this work proposed to use decentralized network for executing desktop applications which developed using of own programming languages and developed to run on complete isolated operating system. It allows to process data faster and without having to have own computing power. Objectives of work are development of new protocols for transfer information, instead of IPv6 and transition to decentralized model the work of services.

Keywords: data processing, algorithm, encryption, hashing, isolated environment, decentralization, database

1 Introduction

In modern world it's impossible to imagine work of computers, laptops, tablets and smartphones without a network connection. At the same time, amount of traffic and data that needs to be stored or processed increases.

Issue of further development of networks and their security is becoming more acute with development of new technologies. Therefore, the issue of information availability, increasing speed of its receipt and use, improving efficiency and quality of computing is very relevant at the present time.

Solution to all of above problems is the decentralization of networks and applications. Therefore, many studies in this area are being carried out and practical applications are already in place [1-3].

2 Formulation of problem

Currently, almost all applications work as client-server services. They allow you to exchange messages, send files, receive news and publish them. Services interact with both users and other services that are often used by intruders when stealing public information, bank card data, private data, files, etc. Therefore, data protection is one of priority concerns for service delivery companies.

Another problem is rapid development of the internet of things. According to latest forecasts in the coming years, the number of devices connected to the Internet will increase several times, so issue of their personal identification is important. According

to growth rate of the Internet of things, the range of IPv6 addresses may end much earlier [3]. Therefore for reliable operation of the Internet these tasks need to be solved: the development of new protocols for the transfer of information and the transition to a decentralized model of service [4-6].

Replacement of services, their decentralized analogues, will solve the following issues: improve the reliability of services itself; increase the speed of services and to make the transition to new protocol for information transferring.

Purpose of work is to implement a system for support of computing in decentralized networks which supports the work of application services. Object of study is the process of implementing a system for support of computing in decentralized networks. Subject is the models, methods, and tools for support of computing in decentralized networks.

The main tasks of work: construction of models and algorithms of work of individual modules of the system; development of decentralized network, which provides data exchange, decentralized storage of information and the launch of services in the network; development of the protocol for transmission and processing of in-formation for decentralized networks of different purposes; implementation of the system for supporting the work of services and conducting its testing on performance [3].

3 Software tools and solutions

Hashing algorithms are common and have several implementations (MD5, SHA, RIPEMD, etc.). The main characteristics of the hashing algorithms are: size; complexity of calculation; cryptographic stability; the speed of the calculation and the number of collisions [7].

Currently, the MD5 algorithm is not crypto-resistant. SHA combines various algorithms such as SHA-1, SHA-2, SHA-3, which are very popular and widely used in the modern world. SHA algorithms are characterized by the fact that making even small changes in input string significantly change the hash. Having looked at the hash algorithms SHA-2 group was chosen because they are protected, sufficiently researched, built on the basis of the structure of the Merkel-Demgard and have a high speed of execution [8-10].

Currently there are many encryption algorithms such as AES, RSA, 3DES, GOST 28147-89 and others. Different decentralized networks use different encryption algorithms. Often you can find the use of two or more algorithms for transmitting and storing information. The most secure and popular encryption algorithms are symmetric - AES, 3DES, Twofish and asymmetric - RSA, Diffie-Hellman, DSA, ECC and ECDH.

There are two encryption algorithms - ECDH for generating a shared key and AES for further encryption of data, which used in the work. This combination gives the system a high degree of stability, since: encryption keys are not transmitted due to unprotected communication lines; AES algorithm is most-protected at the moment; input data for generating a shared key is stable and computed very quickly; there is a possibility to encrypt large volumes of data [11-15].

There are two protocols for information transfer - TCP and UDP. Both protocols work on the transport layer of OSI model. Each protocol has its own peculiarities and

advantages. Sometimes applications support work using both protocols. Advantages of TCP are: reliable delivery of segments; streamlining segments when receiving; work with sessions; transferring speed control.

The advantage of UDP is high speed of transmission relative to TCP. The UDP protocol is used when it is necessary to receive data quickly. When using UDP, a logical connection is not created, so anyone can send data to the application. After reviewing both protocols, it is resolved to use TCP because of its reliability, which is very important in decentralized networks [6].

4 System development

System of computing support is decentralized and overlay. It has high security requirements for data stored and transmitted in system. Since the system is overlay, there is a need for customer identification, so you can use any sequence of characters of any length as the address. This avoids ending the range of available addresses by adding additional characters without changing the entire system.

Reliable encryption algorithms are used to achieve the required security level. Since the AES encryption algorithm, which is a symmetric algorithm, is selected, there is a weak point in the transmission of encryption keys. To solve this, we use an algorithm that generates a shared key without passing it through the network [16].

Using the Elliptic Curve algorithm, customers generate their own private and public keys. Private key is stored by the client and never transmitted by network, and open key is the client's address in a decentralized network. Every client knows the public key of each client with whom the link is established. Using the ECDH algorithm, clients, having their public and private keys and public key of a second client, can generate a shared key that is the same when generating on other side, when using another pair of keys and another public key.

This approach eliminates the need to pass the key through the network and allows you to combine the advantages of symmetric and asymmetric encryption algorithms while eliminating their shortcomings. In this case, the use of a public key as a client's address ensures that the virtual address can’t be mapped to the physical location of the client or Internet. This algorithm generates a public key of large size, which for a long time, if necessary, will ensure a significant expansion of the address range by adding only one character to address.

When communicating information, customers use different operating systems (interaction between smartphone and computer) and data formats to achieve one result. Two methods are used simultaneously to solve these issues: serialization and encapsulation. When serialized, data structures are converted into one common format - text.

Having received text, the client converts it into the necessary structure. For serialization, we use the JSON data format.

```
import sys, os
import json
from serialization import Encoder
sys.path.insert(0, os.path.dirname(os.getcwd()))
class Serializer(object):
```

```
__instance = None
@staticmethod
def get_instance():
    if Serializer.__instance is None:
        Serializer.__instance = Serializer()
    return Serializer.__instance
def __init__(self):
    self.types = {
        # messages types
    }
    self.encoder = Encoder.Encoder()
def serialize(self, obj):
    return json.dumps(obj, default=self.encoder)
def deserialize(self, message):
    return json.loads(message, default=self.encoder)
def get_type(self, obj):
    return obj.__class__.__bases__[0].__name__
def add_type(self, obj, tag):
    obj_type = self.get_type(obj)
    self.types[obj_type] = tag
```

It is quite easy to use and has support in modern programming languages. For serialization of complex data structures, we use encapsulation: data themselves are enclosed in an object that further has a description of these data. With this description, the client will uniquely identify type of data and correctly handle it [17-21].

Since various applications are implemented by different people, and recipient indicates the sender itself, then to solve problems using a single description for different services, the programmer himself should take care of description uniqueness.

For example, the customer sends data that is stored in a separate class.

```
from dc.package import package
class weather (package):
Def __init __ (se lf):
self.country = ''
self.city = ''
def fields (self):
return ['country', 'city']
```

Client pre-creates the association of this class with its code, for example, "weather.request". In the example, the country field contains the value "UA", and the city is "zaporizhzhia". In the serialized form, this class is described below.

```
{"Type": weather.request, "date": {country: UA, city,
zaporizhzhia}}
```

After receiving the data in this form, the client easily reproduces the object of the desired class. The data is encapsulated in a separate object, which contains additional

information that is necessary for the client (the specific data with which the client uses in a separate implementation). The final package view in abbreviated form is given.

```
{"sender": "...", "nodes": ["...", "..."], "length": "...",
"date": {"date": { ... }, "id": "...", " number ": 1, "total": 1,
"crc": ... }}
```

Package has a lot of additional information, so the transmission of small volumes of data occurs at a lower speed. Packets data is encrypted and available only to the recipient and sender, and route information is publicly available [22, 23].

Therefore, anyone can receive the package and send it to the recipient without access to the data themselves. The system model is shown in Fig. 1.

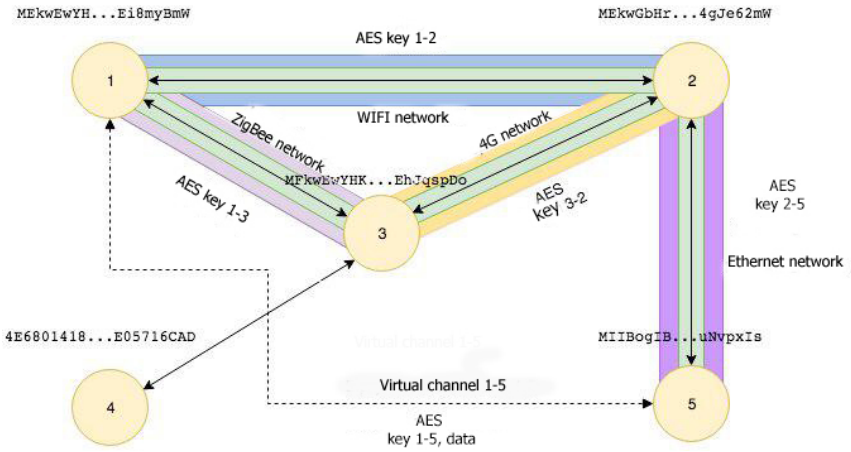

Fig. 1. System model

Important part of system are commands. The command is a regular class that is passed within framework of system and additionally has necessary methods that implement correct behavior.

System work is implemented on commands: routing, launching and publishing applications, receiving data from memory, etc. This makes testing easier and expand functionality and greatly reduces errors in commands and system as a whole [3]. Main purpose of commands - perform certain actions for clients. And the important difference from applications is that they do not work decentralized. To start the command, the client must have an address of another client who will execute this command.

Command can be sent by someone who knows the necessary information on command (data format and its code). Only client who has a registered command with this code can execute the command. In case of receiving a command with the same code but assigned to another client, the behavior is unknown, because the client can't determine if command was sent correctly. With the help of commands, a chain of routing is implemented, applications are published and information from distributed memory is obtained.

For correct delivery of data, client must manually build the necessary chain before transfer. To do this, he performs following: defines client from which to start a search; sends request on presence of the recipient in the list of connected clients; handles the response from the client. If recipient is found, the process is interrupted, otherwise the sender receives a list of all connected customers and begins to search again using another client as a final one. Using this algorithm allows you to control the chain of customers. Client can remove intermediate nodes to increase reliability of system. Implemented chain operates for some time after which it needs to be rebuilt. Therefore, chain changes over time and it makes it more reliable because of physical impossibility of intercepting data in certain areas.

Important part of system is distributed memory that stores all the information that system is operating with, user data and applications itself. Data is stored in memory "as is", so the command or application cares about encryption and decryption of these data. Anyone can get data, but each memory entry has a unique identifier. Therefore, you need to know this identifier to receive data.

Another feature of data in memory is their immutability. After entering data in memory, it becomes impossible to edit them. Distributed memory is a distributed hash table. Physical data is a separate file on storage device, so maximum size of distributed memory depends on number of clients and on how much each client allows to occupy space on data storage device for memory usage. The minimum and maximum data size that can be stored in memory does not exist.

The application in this system is program written using any programming language that is unique in creating decentralized applications. Programs functioning in a decentralized system and on the local computer is no different as they are identical. Applications interact with system as follows: send replies and new data to customers; cause commands; launch new applications; read and write data from distributed memory. Additionally, they can use graphic adapters for complex calculations, make requests to the Internet or work with the file system.

To ensure security of the data, applications are launched in a special, isolated environment. We are using Docker to support running of applications developed in any programming language and different operation systems. Using an isolated environment allows you to separate an application from a physical computer during execution by protecting data and user files.

This approach allows you to run several applications in parallel, provide them with isolated work from each other and from the client operating system. Docker also allows maximum application portability and uses open source technology to easily test applications in a seamless environment before publishing to the system.

Implementation of the system has been performed in form a set of different modules that work together to fulfill the tasks. Using the modular structure allows you to change parts of the system without downtime of system as a whole in cases: transferring the application to another operating system, or not supporting the already existing code in whole or in part [3].

System has the following modules: networks; hashing; encryption; serialization; messages; commands; applications; memory; extensions and routing module. Some modules contain little code so they can been replaced by direct calls to required functions in system body, but process of transferring code to other systems will become more complex and require much more time.

These modules include encryption, hashing, serialization and network modules. Extension module can also been attributed to this set of modules, but it does not integrate into other system modules and should work separately. Other modules are much more complex and perform the basic system functionality. These are message modules, commands, applications, memory, extensions, and routing. Any system module is used in another module or use other modules for calculations or data (Fig. 2).

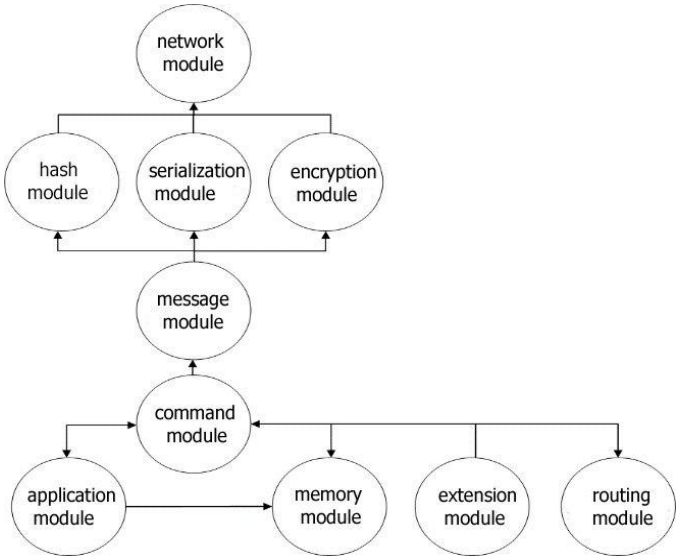

Fig. 2. Interconnection of system modules

Splitting system into separate modules allows you to easily test behavior of modules. Since each separate module is small compared to entire system, its development time and complexity are minimal and this greatly increases the system reliability [3].

System can work in two modes: transit and client. In transit mode, application only transmits the data from sender to recipient, system does not change or receive the data and works only with header. In client mode, application receives data from other users and initializes transfer. It also works simultaneously in transit mode for transmission of messages that are not assigned to current client.

The routing module is one of the most important on network. Algorithm for on-ion routing is chosen routing base. With one difference - not all encryption package, but only the chain address. This approach allows the transfer of information safely and unexpectedly, since the transfer route that is defined by the client and route is securely encrypted.

The chain segments are encrypted the previous segment from end, in the way that a separate key is used for each segment. The closer the segment is to the end, the more times it is encrypted, so the number of encryption can be calculated by the formula (1):

$$N_e = i - 1, \tag{1}$$

where N_e – number of encryption iterations; i – number of current chain segment.

Algorithm of building the chain is given in Fig.3.

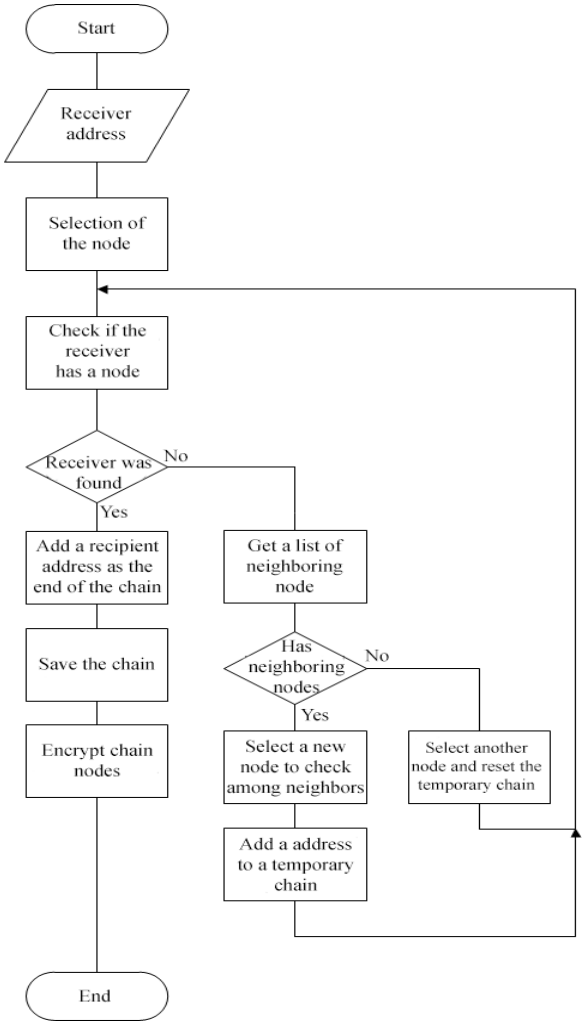

Fig. 3. Algorithm of building a nodes chain for information transmission

For system to work properly, individual modules should only work with module interface. Each module consists of the minimum set of classes. For correct modules operation programmer must implement them correctly. Modules functioning is closely linked to implementation of dependencies. For example, we have hash module interface.

```
from abc import abstractmethod
class hashing ():
```

```
@abstractmethod
def encode (self, object):
passport
```

And also an example of interface implementation.

```
from dc.module.hashing import hashing
from Crypto.Hash import SHA256
class HashingImplementation (Hashing):
def encode (self, object):
try:
h = SHA256.new (data = object)
except ValueError as e:
h = SHA256.new (data = bytes (object, 'utf-8'))
return h.hexdigest ()
```

Dependencies registration and usage example is given below.

```
from dc.injection import inject
from dc.module.hashing import Hashing
from module.hashing import HashingImplementation
inject.get_instance().bind(Hashing, HashingImplementation)
# using:
inject.get_instance().get(Hashing).encode(…)
```

Implementation is completely separate from interface and if there is an error in module it can be corrected without touching other modules and without affecting algorithm of system as a whole.

5 System testing

When connecting to the system, the client must have an IP address already connected to client system. If a connection with client who does not have a connection to other clients occurs, a new isolated network with its own memory, applications, and users is created. After opening a socket and establishing a connection, each client sends own public key to another client. After receiving the key, which is the client's address on the network, a shared key is generated [24].

When sending data, client collects it in a certain class, which represents this data in a virtual form, and receives address of recipient. After filling class, sender initiates the preparation for dispatch. At this time, data is converted into a package – it is serialized and segmented if necessary. After the segmentation, the data is transmitting to the routing module, which defines chain of nodes through which route of their delivery is laid. If the chain is already built and it is relevant, each segment is individually encapsulated into a transit packet and transmitted to network module, which transmits the data.

If the chain is not found, the connected client from the list of clients chooses another client, taking into account the necessary criteria and asks for connection with

the recipient. In case of communication with the recipient, sender adds this client to the list of nodes and decides whether the received chain corresponds to certain criteria. Data is encrypted as required. Segment is encrypted with shared key with the recipient. When receiving a package by a transit client, it reads the chain of nodes, removes itself from chain and sends package further according to route. Once packet is received, the recipient can decrypt it by generating a shared key. After receiving all segments, those segments are merged into one package and deserialized. The following is sent to a command or application, or executed if data is a command.

If it is necessary to start the service, client uses the command, sends the request for an execution to the system closest to client with whom he has a direct connection. This system is a "router" and acts as an intermediate element between client who initiated the launch and the working program. Through this node that exchange of information between the client and the application takes place. Intermediate client copies application from memory and sends a broadcast packet containing an application information request. Upon request, each client decides whether to execute the application or not. If it can, then the client sends answer to "router".

After receiving response, the application is sent to this client, while others will be denied. Client who runs the application runs an isolated environment and copies the application to it. After that, application starts up, provides the necessary service, completes execution and informs the router.

Upon receiving the message about the completion of the application, "router" informs the client and is deleted. While running the application, he can send responses to client requests about the implementation of the application to the "router" and from the "router" to the client. The chain in this case is not being built, and transfer process is the same as during the transfer between different clients.

For testing purpose, an application that executes multiplication of matrices is created, runs in same thread and is written in Java programming language. According to the testing scenario, user received a matrix, which is the result of multiplying two input matrices of 200x200 size. Testing was carried out on local and remote nodes. On the local node - the application was launched in usual environment of operating system, and on the remote node - in an isolated environment. During the testing, 50 matrix multiplications were performed. Matrix elements are eight bytes long.

```
import java.util.Random;
public class Main {
    public static void main(String[] args) {
        (new Main()).run();      }
    private void run() {
        int round = 50;
        int rounds = 100;
        int matrixSize = 200;
        do {
      long[][] matrixA = this.createMatrix(matrixSize, matrixSize);
      long[][] matrixB = this.createMatrix(matrixSize, matrixSize);
           this.fill(matrixA, matrixB, matrixSize);
      long startedAt = System.currentTimeMillis();
      long[][] matrixC = this.multiply(matrixSize,matrixA,matrixB);
```

```
      long endedAt = System.currentTimeMillis();
            System.out.println(endedAt - startedAt);
            round++;
        } while (round < rounds);    }
    private long[][] createMatrix(int n, int m) {
        return new long[n][m];     }
    private void fill(long[][] matrixA, long[][] matrixB, int
matrixSize) {
        Random rand = new Random();
        for (int i = 0; i < matrixSize; i++) {
            for (int j = 0; j < matrixSize; j++) {
                matrixA[i][j] = rand.nextLong();
                matrixB[i][j] = rand.nextLong();
            }        }      }
    private long[][] multiply(int size, long[][] matrixA, long[][]
matrixB) {
    long[][] matrix = this.createMatrix(size, size);
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                for (int k = 0; k < size; k++) {
                    matrix[i][j] += matrixA[i][k] + matrixB[k][j];
                }            }
}
        return matrix;
    }
}
```

To construct the graph, the portion of measurements was used. A non-optimal algorithm was used for testing, since it can clearly be seen the difference in performance of the applications. Results of testing in isolated and normal environments are shown in Fig.4, where: dotted line is an isolated, continuous - usual environment.

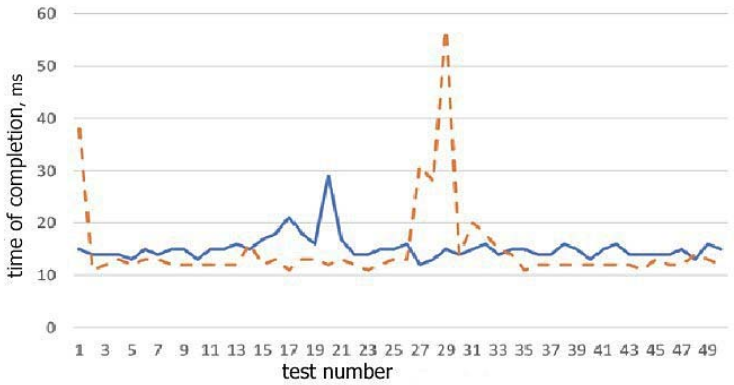

Fig. 4. Test results

Speed of information transfer in the system falls significantly due to some reasons: maximum transmission speed can't exceed the minimum speed between any segment of the transmission chain; there is an additional time for the building of chain; data transmitted by the system is encrypted; additional data is transmitted together with the payload.

During testing, a chain of four segments was used, two of which were intermediate. The following data was obtained: average size of additional data is 222 bytes; size of encrypted data is 256 bytes. It should be noted that through of use of encryption, minimum amount of data transmitted was about 159 bytes, even when transmitting several bytes of data. Thus, when using default MTU on Fast Ethernet (1500) network and TCP protocol, instead of transmitting 1476 bytes of information, it is possible to transfer no more than 1000 bytes (system limitation).

On pair with encryption and additional information, volume of information is 1745 bytes. Thus, the drop of transmission speed is additionally about 30 percent. It should be noted that while transmitting small volumes of information, transmission speed is much smaller, since the significant part of data transmitted is the useless data.

Relatively low transmission speed is compensated by speed of application execution. Comparing overall speed with existing systems is impossible because of their principle of operation, namely because of their strict adherence to data processing schedule and delays in information sending and receiving.

As it can be seen from the results, the execution time of one and the same application is different [24]. Most applications in an isolated environment run faster.

6 Conclusion

In this paper the stages of implementation computing support system in decentralized networks of different purposes is presented. It is proposed to use a decentralized system to run services over an isolated operating system, which makes it possible to process data faster and without the need to have its own computing power.

Proposed system can be used as a typical example in many spheres of human activity: in the legal and financial spheres when automating the signature of contracts as a third independent party; in medicine for formation of single patients database, with further development of the "virtual doctor" module; in the Internet of things, as a platform for operation of devices and their applications, using the transparent interaction of different parts of the system among themselves; in military sphere, as a platform for the automation of machines and safe coordination of military units, using open communication channels.

In the future, it is planned to expand the system by integrating it into mobile devices and other existing Internet services.

References

1. IETF Tools. Mobile Ad hoc Networking (MANET). http://tools.ietf.org/html/rfc2501 (1999). Accessed 03 Feb 2019

2. Murarka, N.: Decentralized and Trustless Networks. https://hackernoon.com /decentralized-and-trustless-networks-f881671fae4e (2018). Accessed 10 Feb 2019
3. Kirichek, G., Rudkovsky, O., Timoshenko, V.: Decentralized computing support system. Scientific notes of TNU named V.I. Vernadsky Series: Engineering. Kyiv, vol. 29 (68), no.6, pp. 161-166 (2018)
4. A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper (2019). Accessed 10 Feb 2019
5. Akhmetov, B., Korchenko, A., Sidenko, V. et al: Applied Cryptology: Encryption Methods. Almaty: KazNRTU (2015)
6. Kirichek, G., Kurai, V.: Implementation quadtree method for comparison of images. In: 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2018, Slavske, pp.129-132 (2018) doi: 10.1109/TCSET.2018.8336171
7. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. https://eprint.iacr.org/2006/105.pdf (2006). Accessed 10 Feb 2019
8. Rivest, R.: The MD5 Message-Digest Algorithm. https://www.ietf.org/rfc/rfc1321.txt (1992). Accessed 10 Feb 2019
9. Wang, X., Feng, D., Lai, X.: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. https://eprint.iacr.org/2004/199.pdf (2004). Accessed 10 Feb 2019
10. Eastlake, D., Jones, P.: US Secure Hash Algorithm 1 (SHA1). https://tools.ietf.org/html/rfc3174 (2001). Accessed 10 Feb 2019
11. Eastlake, D., Hansen, T.: US Secure Hash Algorithms. https://tools.ietf.org/html/rfc4634 (2006). Accessed 10 Feb 2019
12. Announcing the advanced encryption standard (AES). http://www.impic.org/papers/Aes-192-256.pdf (2001). Accessed 10 Feb 2019
13. Karn, P., Metzger, P., Simpson, W.: The ESP Triple DES Transform. https://tools.ietf.org/html/rfc1851 (1995). Accessed 10 Feb 2019
14. Rivest, R., Shamir, A., Ademan, L.: Cryptographic communications system and method. https://people.csail.mit.edu/rivest/pubs/RSA83b.pdf (1983). Accessed 10 Feb 2019
15. Rescorla, E.: Diffie-Hellman Key Agreement Method. https://tools.ietf.org/html/rfc2631 (1999). Accessed 10 Feb 2019
16. Ship, M., Yiqun, L.: Cryptanalysis of Twofish (II). https://www.schneier.com/twofish-analysis-shiho.pdf (1999). Accessed 10 Feb 2019
17. Lafore, R.: Data Structures and Java Algorithms, St. Petersburg: Peter (2013)
18. Deitel, P., Deitel, H.: Visual C# How to Program. Pearson (2016)
19. Burger, W., Burge, M.J.: Digital Image Processing: An Algorithmic Introduction Using Java. Springer (2016)
20. Heineman, D., Pollis, G., Selkov, S.: Algorithms. Reference with examples in C, C ++, Java and Python. Moscow: Alpha-book (2017)
21. Malik D.S.: C++ Programming: From Problem Analysis to Program Design, ISBN-13: 978-1337102087 (2017)
22. Corner, A.: Elliptic Curve Cryptography: a gentle introduction. http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/ (2015). Accessed 10 Feb 2019
23. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-56ar.pdf (2007). Accessed 10 Feb 2019
24. Kirichek, G., Ovchar, A.: The system of data protection at transmission in the network. Scientific notes of Chernovtsy National University. Series: Computer Systems and Components, t.7, vol.2, pp. 32-39 (2016)