УДК 004.7
DOI https://doi.org/10.32838/2663-5941/2021.2-1/36

Рудьковський О.Р.
Донецький національний технічний університет

Киричек Г.Г.
Національний університет «Запорізька політехніка»

ПРОГРАМНИЙ КОМПЛЕКС ІЗ ПІДТРИМКИ РОЗПОДІЛЕНОЇ ВЗАЄМОДІЇ МЕРЕЖЕВИХ ПРИСТРОЇВ ТА ДОДАТКІВ

У роботі запропоновано модель системи та метод програмної реалізації протоколу з підтримки безпечної взаємодії мережних пристроїв та додатків. У процесі реалізації системи й алгоритму її роботи найбільшу увагу приділено безпеці обміну даними і їх обробки. Для різних типів вузлів рекомендується використання різних типів пакетів. При цьому маршрутизатор повинен підтримувати безпеку передачі даних, не маючи до них доступу. Як базову запропоновано використовувати оверлейну мережу, поклавши функції передачі інформації на більш низькі рівні, абстрагувавшись таким чином від мережі передачі даних, операційної системи і технологій, що використовуються. Метою роботи є проведення досліджень та реалізація програмного забезпечення для підтримки безпечної розподіленої взаємодії мережних пристроїв та додатків. Авторами пропонується об'єднати переваги мережі Ethereum та хмарних сервісів, у результаті чого розробники отримають можливість створювати додатки, використовуючи будь-яку мову програмування. При цьому відстань до серверів значно зменшиться, дані будуть надійно зашифровані та доступні лише тому, хто їх створив. Для досягнення основної мети пропонується реалізувати: метод маршрутизації клієнтів у системі, який дозволить прокладати максимально швидкий та безпечний маршрут; алгоритм запуску довільного коду в межах системи, що дасть змогу запускати будь-який код на будь-якому кінцевому вузлі максимально безпечно для користувача; метод зберігання та обміну інформацією, яка передається від додатка до додатка та від пристрою до пристрою, зберігаючись тривалий час, із можливістю отримання її у будь-який момент, а також дослідити функціонування додатків і пристроїв, спираючись на реалізовані алгоритми та методи. Використання ізольованого середовища робить процес запуску додатків відокремленим від операційної системи та інших програм, які можуть перехоплювати дані.

Ключові слова: *мережа, алгоритм шифрування, додаток, пакет, адресація, маршрутизація.*

Постановка проблеми. Нині майже всі додатки працюють у вигляді клієнт-серверних сервісів та дають змогу обмінюватися повідомленнями, передавати файли, отримувати новини та публікувати їх. Але сервіси не є достатньо захищеними. Особливо це стосується даних банківських карток, приватної інформації різного призначення тощо, тому їх захист є пріоритетним [1]. Виходячи з цього, вже зараз треба вирішити низку актуальних питань, пов'язаних із надійною роботою мереж у майбутньому, таких як: розроблення нових протоколів передачі інформації; перехід на децентралізовану модель роботи сервісів; збільшення їх надійності та швидкості роботи [2].

Аналіз останніх досліджень та публікацій. У сучасному світі вже є системи, що здатні частково виконувати поставлені завдання. Найбільш розповсюдженою та функціональною є Ethereum. Вона використовує спеціальну мову програмування Solidity та віртуальну машину – EVM. Функціонування додатків, написаних цією мовою, є складним, оскільки мова програмування і віртуальна машина накладають певні обмеження [3].

Для запуску додатків розробники також часто використовують і хмарні обчислення, які дають змогу швидко розгортати їх нові екземпляри. До компаній, які надають такі можливості, належать Amazon, Google та Microsoft. Але використання хмарних серверів не гарантує достатнього захисту тих даних, які на них зберігаються [4]. Слід зазначити, що більшість підходів, пов'язаних вз роботою сервісів в Інтернеті, нині мають низку невирішених питань. Для традиційних систем це питання безпеки зберігання даних і передачі інформації. За широкого використання різних методів шифрування вони, наприклад, повноцінно не захищають від атак виду man in the middle (MITM) чи проникнення на сервер, де дані зазвичай вже не є шифрованими [5]. Також розподілені системи мають недоліки, пов'язані зі швидкістю

роботи та функціональністю сервісів. У мережі Ethereum завдання обробляються «коли-небудь», функціональність обмежена. У неї немає функцій підтримки роботи з числами, мережними даними, файлами тощо.

Крім цього, для зберігання інформації у розподілених системах використовують блокчейн. Він гарантує незмінність, відкритість даних та безпеку їх зберігання. Проте блокчейн уразливий до атак типу 50%+1, коли кількість ресурсів впливає на роботу всієї системи [6].

Виходячи з цього, авторами пропонується об'єднати переваги мережі Ethereum та хмарних сервісів, у результаті чого розробники отримають можливість створювати додатки, використовуючи будь-яку мову програмування. При цьому відстань до серверів значно зменшиться, дані будуть надійно зашифровані та доступні лише тому, хто їх створив.

Постановка завдання. Метою роботи є проведення досліджень та реалізація програмного забезпечення для підтримки безпечної розподіленої взаємодії мережних пристроїв та додатків. Об'єктом дослідження є процес реалізації системи підтримки розподіленої взаємодії мережних пристроїв та додатків. Предметом дослідження є моделі, методи та програмні засоби організації безпечної взаємодії пристроїв та додатків у розподіленому середовищі. Для досягнення основної мети авторами пропонується реалізувати: метод маршрутизації клієнтів у системі, який дозволить прокладати максимально швидкий та безпечний маршрут; алгоритм запуску довільного коду в рамках системи, що дасть змогу запускати будь-який код на будь-якому кінцевому вузлі максимально безпечно для користувача; метод зберігання та обміну інформацією, яка передається від додатка до додатка та від пристрою до пристрою, зберігаючись тривалий час, із можливістю отримання її у будь-який момент, а також дослідити функціонування додатків і пристроїв, спираючись на реалізовані алгоритми та методи [7–8].

Виклад основного матеріалу. Архітектура системи схожа на звичайну архітектуру «клієнт – сервер». Система є розподіленою, кожний користувач взаємодіє з нею, використовуючи один або декілька з доступних вузлів. Хоча вузли і взаємодіють один з одним напряму, їхня робота не залежить один від одного. Вузол може підключитися до будь-якого іншого вузла і тим самим створювати новий потенційний маршрут або декілька альтернативних маршрутів. Важливим моментом у системі є підтримка роботи пристроїв, тому спроектовано відповідну модель (рис. 1).

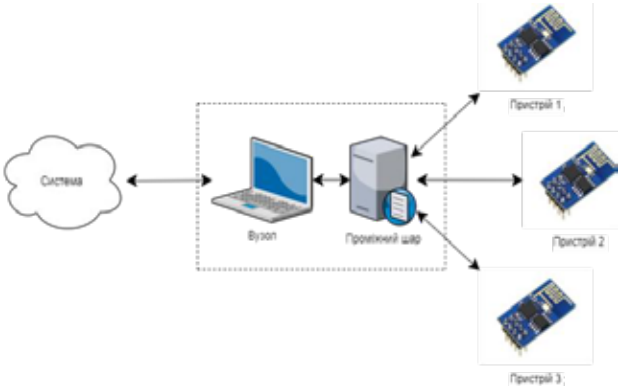

Рис. 1. Модель взаємодії пристроїв

Проміжний шар використовуємо виходячи з того, що пристрої самостійно не зможуть шифрувати дані, підраховувати хеші та приймати складні рішення так швидко, як це необхідно для їхньої роботи. Тому ці функції переносяться на найближчий захищений вузол локальної мережі [9]. Розглянувши моделі взаємодії клієнтів, вузлів та пристроїв, перейдемо до моделювання архітектури додатку [10]. Виділяємо такі загальні модулі: модуль підключень – встановлює підключення між учасниками системи за допомогою необхідного протоколу зв'язку; вхідних повідомлень – опрацьовує вхідні повідомлення по окремому підключенню; обробки пакетів – виконує обробку пакетів незалежно від підключення; шлюз – для передачі інформації іншому учаснику системи, до якого є відкрите підключення; виконавчий – для запуску додатків; проміжний шар – зв'язує пристрої та систему; підсистема пам'яті – зберігає та зчитує інформацію, яка є у системі чи на поточному вузлі; програмний проміжний шар – зв'язує ізольовані додатки з системою. При цьому система повинна працювати з кінцевими вузлами (клієнтами), пристроями та додатками у ізольованому середовищі. Оскільки пристроям та додаткам не треба передавати всю інформацію, яка необхідна для роботи системи загалом, для них використовується окремий формат даних.

Існує два підходи до створення ізольованого середовища. Перший і найбільш простий – використання віртуальних машин. Другий підхід – логічне відокремлення систем одна від одної. Таким чином, слід виконати низку кроків: створити образ (image), створити контейнер (container) та запустити додаток у ньому. Працює все це через групування процесів. Кожному ізольованому середовищу назначається унікальний номер. Процес у цьому середовищі має такий же номер, як і саме середовище. Для створення

образів та контейнерів є багато команд, але існує проміжний додаток – Docker, який робить все за розробників та адміністраторів. Всі образи створено для нього, тому він може запустити будь-який із них [11].

Особливість контейнерів – їх незмінність та гарантія роботи. Все, що працює у контейнері, втрачається після завершення роботи контейнера. Якщо образ працює на кінцевому вузлі, то він гарантовано працює і на іншому вузлі, який підтримує роботу контейнерів. Тому використання технології контейнеризації та Docker є безальтернативною для реалізації цього програмного засобу. Після встановлення Docker слід закрити та відкрити поточний термінал (якщо встановлення за допомогою SSH) або вийти та увійти у систему. Далі слід перевірити працездатність Docker, виконавши потрібні команди. Обов'язково у останній команді слід замінити your-user на ім'я користувача, який використовує Docker. Ця команда виконує завантаження тестового образу з мережі та запускає контейнер [11].

У процесі реалізації програмного комплексу використано IDEA Intellij IDEA, а для проекту вибрано систему Gradle. Після створення проекту IntelliJ IDEA автоматично запускає Gradle. Він створює скелет додатку, після успішної генерації якого створюємо пакет «ua.edu.zntu.arudkovskiy.yaddro». У цьому пакеті реалізовано усі необхідні класи та інтерфейси, які виконують всі основні функції системи [12]. Цей програмний комплекс є одночасно і вузлом, і проміжним забезпеченням для пристроїв і додатків. Весь комплекс можна поділити на окремі функціональні частини: робота з мережею та протокол передачі інформації; базові функції системи та система команд; підсистема роботи з ізольованим середовищем (контейнерами); підсистеми роботи з додатками та підсистема роботи з пристроями.

Оскільки система є модульною, то слід розділити всі частини на окремі незалежні елементи, які взаємодіють лише з інтерфейсами інших модулів. Так, кожна частина (клас) системи має інтерфейс, що описує функції, які виконує ця частина (клас), та клас, який реалізує інтерфейс. Для зручності окремі класи згруповано у пакети. У структурі проекту виділяємо директорії: src/main, що містить код програмного засобу та src/test, у якій розміщуються тести (рис. 2).

Мережева взаємодія є однією з найважливіших частин програмного комплексу. Для її реалізації використовуємо транспортний протокол Transmission Control Protocol (TCP) та формат

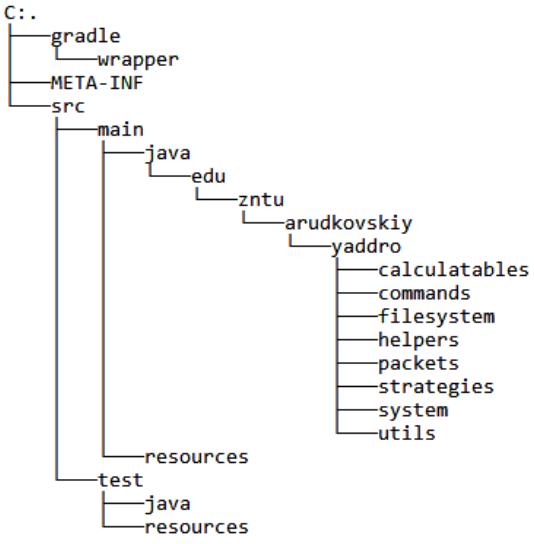

Рис. 2. Структура проекту

даних JSON. Оскільки Java не підтримує цей формат даних, додаємо сторонню бібліотеку Gson. Основним класом у підсистемі мережі є клас Connection. Цей клас виконує підключення до іншого кінцевого вузла та ініціює прослуховування порту. Він містить такі методи: bind(int port) – початок прослуховування порту; connect (String address, int port) – виконання підключення до іншого кінцевого вузла; fromSocket (Socket socket) – перетворення сокету Java на об'єкт типу Connection; transferMyId – відправлення ідентифікатору на іншу сторону; listen – початок прослуховування підключення. Основними методами є bind та connect. Метод bind створює об'єкт типу ServerSocket для нового підключення. Отримавши нове підключення, система створює об'єкт типу Connection, використовуючи метод fromSocket. Оскільки програма працює у одному потоці, а підключень може існувати безліч, під час кожного прослуховування нового порту створюється окремий потік. Крім того, під час кожного підключення до іншого вузла створюється окремий потік, у якому зчитуються дані. Основна робота сервера виконується у окремому потоці [13].

Після запуску сервера й отримання підключення додаток чекає на команду «ready» і приступає до роботи, використовуючи команди: setData – для зміни стану додатка; getData – отримання стану додатка; sendToClient – надіслати клієнту дані; getFile – отримати файл із розподіленої пам'яті; exportState – експортувати стан додатку; runForResult – запустити інший додаток та отримати результат. Додаток отримує такі команди від системи: onData – дані отримано; onFile – файл зчитано; onResult – результат

отримано; onStateExported – стан експортовано; onExternalData – отримані зовнішні дані.

Тестування системи. У роботі для тестування системи реалізовано додаток, який підраховує факторіал великих чисел, приймаючи як аргумент максимальне число. У циклі, поки деяка змінна менша за максимальне число, розраховується факторіал цього числа. Тестування проводилося на вузлі у віртуальній машині. Для неї виділено 4 ГБ оперативної пам'яті та 4 віртуальних ядра. Як операційна системиа використовується Debian із графічним інтерфейсом Gnome. Усі заміри швидкодії проводилися лише у цій віртуальній машині, тому вони є об'єктивними. Для тестування розроблено спеціальний додаток, який опубліковано у системі; додаток запущено декілька разів у різних конфігураціях: у ізольованому середовищі; в ізольованому середовищі з використанням системи та без ізольованого середовища. Для кожного із сценаріїв підраховано середній час обчислення факторіалу (рис. 3):

```
package edu.zntu.arudkovskiy.app_for_testing;
import ua.edu.zntu.arudkovskiy.yaddro.api.YaddroApi;
public class Main {        private int x;
public static void main(String[] args) {
Main main = new Main();
main.x = Integer.parseInt(args[0]);
YaddroApi yaddroApi = YaddroApi.getInstance();
main.doSomething();      }
private void doSomething() {
for (int i = 1; i <= x; i++) {
calculateFor(i);
}      }
private void calculateFor(int number) {
long time1 = System.nanoTime();
long factorial = factorial(number);
long time2 = System.nanoTime();
System.out.println(time2 – time1);     }
public static long factorial(int number) {
long result = 1;
for (int factor = 2; factor <= number; factor++) {
result *= factor;
}
return result;          }
}
```

Рис. 3. Обчислення факторіалу

Максимальним числом, для якого розраховувався факторіал, є 40. Перші 20 результатів для дослідження не враховуються, оскільки виконуються операції над відносно малими числами та є особливості роботи додатків на мові Java [14]. За результатами тестування побудовано графік швидкості виконання додатку (рис. 4). На графіку ось X – число, для якого виконується розрахунок, а Y – час у наносекундах. Оскільки віртуальній машині виділено достатньо ресурсів і операція для тестування не є досить складною, то і час виконання є малим. Результат роботи без системи і без ізольованого середовища є одночасно еталонним для даної віртуальної машини і можливим для запуску додатку на хмарному сервері.

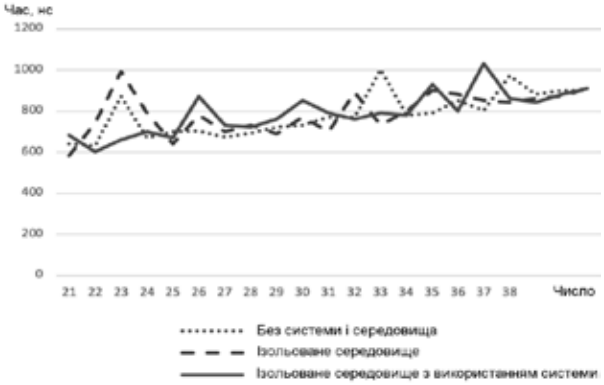

Рис. 4. Порівняння швидкості роботи

Як видно з графіку, час обробки однакової задачі у реалізованій системі та за використання традиційних інструментів має особливості. Система надає такі ж самі можливості, як і традиційні системи, і час виконання додатків у системі практично не відрізняється від часу виконання у традиційних системах, але самі додатки працюють у захищеному середовищі з високим рівнем захисту. Для перевірки правильності роботи з мережею використано програмне забезпечення WireShark. У цьому додатку перехоплено увесь трафік, що проходив через певний порт, перехоплено ключові пакети та перевірено їх коректність (рис. 5). Спочатку – обмін ідентифікаторами, а далі досліджено пакет запиту на виконання:

У цьому методі спочатку створюється необхідна команда, заповнюється ідентифікатор додатка, вказуються аргументи командного рядка та відправляється запит за допомогою іншої команди (ResolveChainAndSend), яка спочатку будує ланцюг, опитуючи вузли рекурсивно, і вже потім надсилає необхідному команду.

Висновки. У роботі наведено модель системи та вдосконалено процес взаємодії мережних пристроїв та додатків у розподілених системах. Реалізовано методи взаємодії окремих частин системи та компонентів додатків. Застосування методів шифрування і передачі інформації підвищують безпеку передачі даних за рахунок багатошарового шифрування даних, а використання ізольованого середовища робить процес запуску додатків

```
        RunApplication runApplication = new RunApplication();
        runApplication.appId                                                    =
"bad42fbfdb131aab1e61cc7ac7dea4b3bc5ec044fe94e4d511c71e76ba49e40f4048d
eee655bfb3b32304e91fee98eda4b145e839a21d1d344e7a28ca412b33f";
        runApplication.args = new String[] { "40" };
        runApplication
        .setOnMessage(message -> System.out.println("Message: " + message))
        .setOnResponse(response -> { /* ... */ })
        .setOnEvent((event, eventArgs) -> { /* ... */ });

        try {
        (new ResolveChainAndSend())
        .setPayload(runApplication)
        .setTargetId("node1")
        .setRaw(new byte[0])
        .execute();
        } catch (Exception e) {
        e.printStackTrace();
        }
```

Рис. 5. Перехоплення трафіку

відокремленим від операційної системи та інших програм, які можуть заважати роботі або перехоплювати дані. Використання модульного автоматичного тестування дає змогу бути впевненим, що усі частини працюютьтак, як треба, навіть після невеликих змін, які могли вивести певний модуль і всю систему з ладу. У подальшому планується розширення можливостей роботи системи шляхом підтримки графічних додатків і розпаралелювання процесів на пристроях мережі.

Список літератури:

1. Corson S., Macker J., RFC2501: Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations, 1999, 12 p.
2. Rivest R.L., Shamir A., Adleman L.M., U.S. Patent No. 4,405,829. Washington, DC: U.S. Patent and Trademark Office, 1983, available at: https://patents.google.com/patent/US4405829/en.
3. Dannen C., Introducing Ethereum and solidity, vol. 1, Berkeley: Apress, 2017, 181 p., doi:10.1007/978-1-4842-2535-6.
4. Jansen W.A., Grance T. Guidelines on security and privacy in public cloud computing, NIST Special Publication 800-144, 2011, 70 p., available at: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecial publication800-144.pdf.
5. Rahim R., Man-in-the-middle-attack prevention using interlock protocol method. ARPN J. Eng. Appl. Sci, 12(22), 2017, pp. 6483–6487.
6. Kolesnikov P., Beketnova Y., Krylov G., Blockchain technology. Analysis of attacks, defense strategies, LAP Lambert Academic Publishing, 2017, 67 p.
7. Kirichek G., Tymoshenko V., Rudkovskyi O., Hrushko S., Decentralized System for Run Services, in CMIS-2019, 2019, pp. 860–872.
8. Kirichek G., Kyrychek D., Hrushko S., Timenko A., Implementation the Protection Method of Data Transmission in Network, in: International Conference on Advanced Trends in Information Theory (ATIT), IEEE, 2019, pp. 129–132.
9. Corbellini A., Elliptic curve cryptography: a gentle introduction, 2015, available at: http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/.
10. Киричек, Г.Г., Киричек, О.О., Модель системи оцінки ідентичності програмного коду, Науковий вісник Чернівецького національного університету, Комп'ютерні системи та компоненти, 2(3), 2011, С. 10–17.
11. Pathania N., Setting up jenkins on docker and cloud, in: Pro Continuous Delivery. Apress, Berkeley, CA, 2017, pp. 115–143.
12. Krochmalski J., IntelliJ IDEA Essentials. Packt Publishing Ltd, 2014, 288 p.
13. Kirichek G., Harkusha V., Timenko A., Kulykovska N., System for detecting network anomalies using a hybrid of an uncontrolled and controlled neural network. In CS&SE@ SW 2019, 2019, pp. 138–148.

14. Bose S., Mukherjee M., Kundu A., BanerjeeM., A comparative study: java vs kotlin programming in android application development, International Journal of Advanced Research in Computer Science, 9(3), 2018, pp. 41–45.

Rudkovskyi O.R., Kyrychek H.H. SOFTWARE PACKAGE TO SUPPORT DISTRIBUTED INTERACTION OF NETWORK DEVICES AND APPLICATIONS

The paper proposes a system model and a method of software implementation of the protocol to support secure interaction of network devices and applications. In the process of implementing the system and the algorithm of its operation, the greatest attention is paid to the security of data exchange and processing. It is recommended to use different types of packages for different types of nodes. In this case, the router must maintain the security of data transmission without having access to them. As a basic, it is proposed to use an overlay network, putting the functions of information transfer at lower levels, thus abstracting from the data network, operating system and technologies used. The paper proposes to combine the advantages Ethereum network and cloud services. As a result, developers will be able to create applications using any programming language. This will significantly reduce the distance to the servers and the data will be securely encrypted and available only to those who created them. To achieve distributed interaction between applications and devices is proposed: to implement methods of routing clients in the system, which will allow to lay the fastest and safest route; to improve methods of encrypting information in the system, which will encrypt data so as to ensure their security and inaccessibility in case of interception by attackers; to implement a method and algorithm for running arbitrary code within the system, which will allow you to run any code on any computer as safely as possible for the user on whose de-vice this application runs; to implement methods and algorithms for storing and exchanging information in the system, which are transmitted from application to application and from device to device and stored for a long time with the possibility of obtaining them at any time; to investigate the algorithms of functioning applications and devices, the basis which are implemented algorithms and methods. The use of an isolated environment makes the process of launching applications separate from the operating system and other programs that may intercept data.

Key words: *network, encryption algorithm, application, packet, addressing, routing.*