|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  Кафедра програмних систем і технологій  **Курсова робота**  на тему  “Розробка блокчейн вузла у якості автоматизованої системи безготівкових платежів” | | | |
| **Виконав:** | Гоша Давід | **Перевірив**: |  |
| Група | ІПЗ-33 | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2022 | | | |

# Анотація

Ця дослідницька робота містить вступне дослідження фундаментальних принципів і концепцій, що лежать в основі біткоїна і технології блокчейн, яка його підтримує, перш ніж заглибитися у всебічний аналіз відомих кібератак, спрямованих на мережу біткоїна. Потім послідовно розглядаються основні компоненти і мережева архітектура кількох відомих блокчейн-проектів, пропонуючи детальний опис їхньої внутрішньої роботи.

Далі в роботі представлено математичну модель мережі, що полегшує поглиблене вивчення її основоположних принципів. Спираючись на ці знання, робота завершується розробкою та пропозицією нової автоматизованої системи безготівкових платежів. Цей синтез теоретичного розуміння і практичного застосування має на меті зробити внесок у дискусію про потенціал і виклики технології блокчейн.

## **Зміст**

ЗМІСТ

[Анотація 2](#_Toc135512908)

[**Зміст** 0](#_Toc135512909)

[**Список малюнків** 0](#_Toc135512910)

[Вступ 1](#_Toc135512911)

[**1.1** **Цілі** 1](#_Toc135512912)

[**1.2** **Практичне застосування блокчейн технології в автоматизованих платіжних системах** 2](#_Toc135512913)

[**1.3** **Порівняння аналогів** 3](#_Toc135512914)

[**1.4** **Короткий зміст** 4](#_Toc135512915)

[Основні поняття 1](#_Toc135512916)

[**2.1** **Баланс** 1](#_Toc135512917)

[**2.2** **Транзакції та UTXO модель** 2](#_Toc135512918)

[**2.3** **Конфіденційність** 3](#_Toc135512919)

[**2.4** **Асиметричне шифрування** 4](#_Toc135512920)

[**2.5** **Proof-Of-Work** 5](#_Toc135512921)

[**2.6** **Proof-Of-Elasped-Time** 6](#_Toc135512922)

[**2.7** **Майнінг** 7](#_Toc135512923)

[Форування блоків 1](#_Toc135512924)

[Топологія мережі, Gossip протокол та її складові 1](#_Toc135512925)

[Доказ парадоксу Пуассона для PoW 1](#_Toc135512926)

[**5.1** **Гіпотеза** 1](#_Toc135512927)

[**5.2** **Технічне завдання на розроблення моделі** 1](#_Toc135512928)

[**5.3** **Математичний опис моделі** 2](#_Toc135512929)

[**5.4** **Аналіз результатів моделі** 5](#_Toc135512930)

[**5.5** **Висновок** 8](#_Toc135512931)

[Реалізація програмного забезпечення 1](#_Toc135512932)

[**6.1** **Створення програмного продукту** 1](#_Toc135512933)

[**6.2** **Класи та об'єкти** 1](#_Toc135512934)

[**6.3** **Реалізація інтерфейсу користувача** 2](#_Toc135512935)

[**6.4** **Реалізація програмного модуля** 2](#_Toc135512936)

[**6.5** **Інструкція для користувача програми** 2](#_Toc135512937)

[**6.6** **Детальний опис класів та об'єктів** 3](#_Toc135512938)

## **Список малюнків**

[Рисунок 1 Специфікація транзакції у банківській мережі 3](#_Toc135415711)

[Рисунок 2 Порівняння вихідного коду net.cpp (ліворуч Dogecoin, праворуч Bitcoin) 4](file:///C:\Users\admin\Desktop\BlockChain\Документ4.docx#_Toc135415712)

[Рисунок 3 Спрощений ланцюжок транзакцій 2](#_Toc135415713)

[Рисунок 4 Специфікація UTXO транзакції 3](file:///C:\Users\admin\Desktop\BlockChain\Документ4.docx#_Toc135415714)

[Рисунок 5 Специфікація взаємодії двох суб'єктів у явно ворожому середовищі 4](#_Toc135415715)

[Рисунок 6 Графічне представлення комп'ютерних мереж 2](#_Toc135415716)

[Рисунок 7 Графічне представлення ієрархії комп'ютерних мереж 3](#_Toc135415717)

[Рисунок 8 Процедура обміну блоком 5](#_Toc135415718)

[Рисунок 9 Специфікація поширення блоку у мережі біткоїну 6](#_Toc135415719)

[Рисунок 10 Графік щільності часу в залежності від кількості транзакцій 5](file:///C:\Users\admin\Desktop\BlockChain\Документ4.docx#_Toc135415720)

[Рисунок 11 Діаграма кількості транзакцій в залежності від часу підтвердження блоку 5](file:///C:\Users\admin\Desktop\BlockChain\Документ4.docx#_Toc135415721)

[Рисунок 12 Середня кількість транзакцій на бло 6](file:///C:\Users\admin\Desktop\BlockChain\Документ4.docx#_Toc135415722)

[Рисунок 13 Графік щільності часу в залежності від кількості транзакцій 7](#_Toc135415723)

**РОЗДІЛ 1**

# Вступ

### **Цілі**

Зважаючи на зростаючу увагу до технології біткоїна серед провідних банківських установ, всебічний аналіз її системи та безпеки має першорядне значення. Однак проведення експериментів на реальній мережі біткоїнів є складним завданням через її незмінну природу. Для подолання цієї проблеми було розроблено симулятор мережі біткоїнів з відкритим вихідним кодом [15], який дозволяє дослідникам отримати уявлення про рівень безпеки мережі та зрозуміти складність системи. Цей проект має на меті використати цей симулятор для вивчення мережі Bitcoin P2P, її комунікаційних протоколів, механізмів безпеки, вразливостей, а також змоделювати процес надходження блоків з точки зору системи масового обслуговування (черги).

Конкретні завдання цього проекту полягають у наступному:

1. Проаналізувати структуру та роботу мережі Bitcoin P2P.
2. Зрозуміти протоколи зв'язку та системи безпеки, що використовуються в мережі Біткоїн.
3. Виявити та оцінити вразливості P2P-мережі для забезпечення комплексної оцінки безпеки.
4. Змоделюйте явище надходження блоків як систему черг, щоб отримати унікальний погляд на функціональність мережі.
5. Розробити дизайн, який інкапсулює вищезгаданий аналіз і виділяє ключові компоненти.
6. Порівняти та зіставити переваги та недоліки нашого проекту з основними конкурентами на ринку на основі результатів аналізу.
7. Зробити глибокі висновки на основі результатів проекту.
8. Кожне завдання сприяє глибокому розумінню мережі Біткоїн та її аспектів безпеки, що полегшує розробку надійних і безпечних мережевих систем P2P.

### **Практичне застосування блокчейн технології в автоматизованих платіжних системах**

Розуміння обмежень і застарілих механізмів найпоширеніших у світі систем грошових транзакцій має вирішальне значення. У ньому висвітлюються переваги, які може принести технологія блокчейн. Домінуючі у світі системи - банківські платежі, що здійснюються переважно за допомогою карток Visa, Mastercard і Maestro - передбачають участь кількох посередників, кожен з яких виконує певну функцію в економіці банківського сектору. Цей багаторівневий процес часто є причиною того, що транзакція в мережі Ethereum проходить швидше, ніж традиційна банківська транзакція.

Розглянемо абстрактний сценарій за участю покупця і продавця, щоб зрозуміти весь ланцюжок транзакцій. Припустимо, що продавець - це ТОВ, і у нього є рахунок в банку Б, який також надає платіжний термінал. Покупець здійснює покупку у ТОВ, прикладаючи банківську картку до терміналу. Ця дія запускає низку запрограмованих кроків, які завершуються формуванням транзакції.

Простіше кажучи, ця транзакція передбачає, що користувач банку "А" має намір переказати 100 одиниць певної валюти до банку продавця "Б", вказавши декілька реквізитів та код категорії платника (КОП). Термінал надсилає дані про транзакцію до банку А. Якщо транзакція є міжнародною, вона повинна пройти через міжнародну платіжну систему, таку як VISA. Після цього VISA надсилає запит до банку клієнта для підтвердження наявності коштів і статусу покупця.

Після виконання умов транзакції банк А надсилає відповідь за тим самим ланцюжком: Банк А - МПС - Банк Б - Термінал. Однак, незважаючи на цей складний процес, торговець отримує гроші лише через три дні.

Цей складний, тривалий процес і покликана спростити та прискорити технологія блокчейн. Зменшуючи кількість посередників і використовуючи децентралізовану, безпечну і прозору систему, технологія блокчейн може забезпечити більш швидкі та ефективні транзакції, що є очевидною перевагою для автоматизованих платіжних систем.

Нижче наведено схему специфікації еквайрингу, де Банк Б є еквайром, а Банк А - емітентом картки:

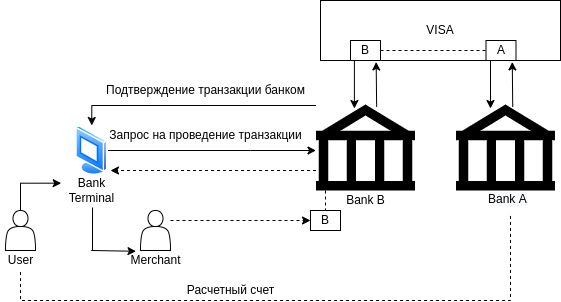


Рисунок 1 Специфікація транзакції у банківській мережі

Технологія блокчейн є інноваційним рішенням для вирішення проблем існуючих автоматизованих платіжних систем, пропонуючи швидкість, безпеку та прозорість.

### **Порівняння аналогів**

На сьогоднішній день існує безліч робіт, присвячених аналізу мережі Біткоїн. Ці дослідження показують, що мережа Bitcoin являє собою складний сценарій з багатьма учасниками, різноманітними інтересами і дуже специфічним методом поширення інформації серед усіх користувачів. З точки зору зловмисника, існує крихка рівновага між наміром експлуатувати і обманювати цю мережу і необхідністю підтримувати безпеку для збереження високої довіри користувачів, що безпосередньо впливає на ціну біткоїна.

Варто зазначити, що кілька інших криптовалют використовують той самий вихідний код для мережевої та P2P-комунікації, що й біткоїн. Ця реальність розширює сферу інтересів, оскільки будь-який недолік безпеки в системі Bitcoin може потенційно вплинути на всі інші криптовалюти, що використовують той самий код.

Відомо, що наступні криптовалюти мають схожий з біткоїном вихідний код для P2P-мережі:

* Litecoin: https://github.com/litecoin-project/litecoin
* Dogecoin: https://github.com/dogecoin/dogecoin
* Ethereum: https://github.com/ethereum/go-ethereum
* Solana: https://github.com/solana-labs/solana
* Cardano: https://github.com/input-output-hk/cardano-node
* Polygon: https://github.com/maticnetwork/matic-docs

Було помічено, що деякі з цих криптовалют були розроблені шляхом створення форку останньої версії вихідного коду біткоїна з подальшими модифікаціями для адаптації його як нової криптовалюти. Однак основна структура коду залишається вкоріненою у вихідному коді біткойна.

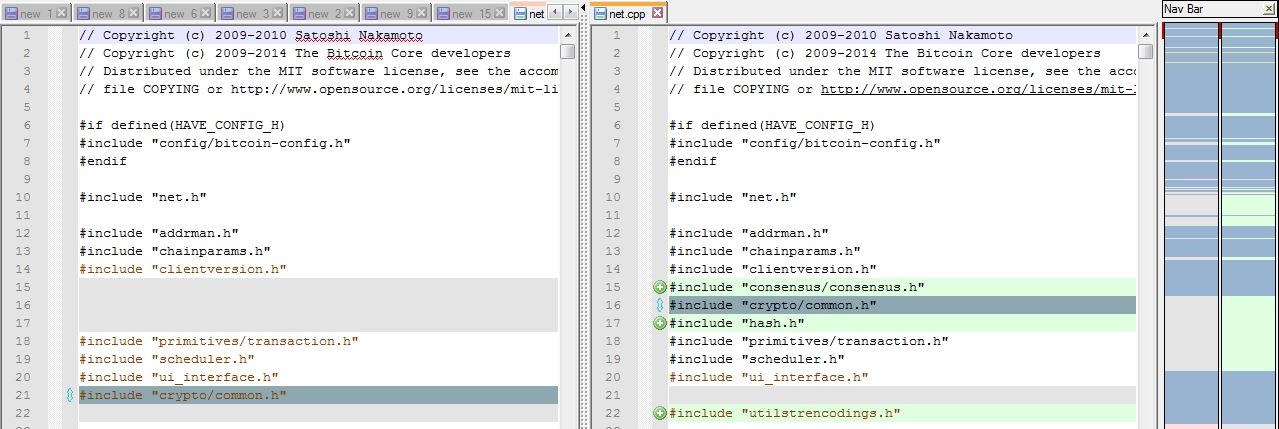
Dogecoin слугує наочним прикладом цього. Як показано на рисунку 1.2, розробники використали вихідний код біткоїна як основу, застосувавши зміни для розробки своєї криптовалюти відповідно до своїх вимог. Хоча ці зміни потенційно можуть усунути певні вразливості в коді біткоїна, все ж таки вкрай важливо виявити будь-які можливі проблеми в мережі біткоїна для їх усунення або оцінки впливу на інші криптовалюти.

Рисунок 2 Порівняння вихідного коду net.cpp (ліворуч Dogecoin, праворуч Bitcoin)

Існує обмежена кількість комплексних рішень у сфері симуляції біткойна. Найбільш релевантними роботами в цій галузі є симулятор Python [29] та Simbit [12]. Симулятор Python спеціально розроблений для протоколу Bitcoin, однак йому не вистачає можливості демонструвати детальні дані моделювання.

І навпаки, Simbit є загальним мережевим симулятором, який включає бібліотеку для симуляції клієнта Bitcoin, але його розробка ще триває. Враховуючи цей контекст, ми вважаємо, що оптимальним підходом для розробки цього проекту є використання симулятора мережі Bitcoin [25], розробленого Віктором Морою як фінальний проект для MISTIC. Цей симулятор був створений командою UAB [15] за підтримки Bitcoin Foundation. Протягом останніх місяців команда UAB займалася розробкою різних модулів програмного забезпечення симулятора. Цим проектом ми прагнемо доповнити симулятор необхідною функціональністю для видобутку відповідних даних, їх обробки та представлення у форматі, зрозумілому для користувачів.

### **Короткий зміст**

Зміст цього документа впорядковано за розділами таким чином:

* Розділ 1: Ознайомлення з проблемами та цілями роботи.
* Розділ 2: Основні поняття, які фігуруватимуть у роботі.
* Розділ 3: Формування боків та їх складова.
* Розділ 4: Топологія мережі, Gossip протокол, та її складові.
* Розділ 5: Доказ парадоксу Пуассона для біткоїну

**РОЗДІЛ 2**

# Основні поняття

У цьому розділі ми представимо деякі фундаментальні визначення, які використовуються в цьому документі щодо біткоїна та відомих атак на його мережу.

### **Баланс**

Топологія мережі, з якою ми маємо справу, є децентралізованою одноранговою, оскільки приклад, який ми розглядаємо, - це Біткоїн. Така мережева архітектура означає відсутність центральної точки управління. Така технологія породжує певні неоднозначні проблеми, такі як подвійні витрати. Проблема подвійних витрат була вирішена за допомогою ланцюжка цифрових підписів. Щоб зрозуміти це, нам потрібно абстрагуватися від традиційної моделі обміну фіатних грошей.

Біткоїн не має поля "баланс" або фізичних монет як таких. Існує лише один реєстр, який є ланцюжком усіх транзакцій. Звідси ми можемо математично визначити баланс вузла. По суті, це можна представити як ланцюжок передачі прав власності на частину загальної емісії валюти. Самі монети знаходяться в стані мономорфності і існують виключно для спрощення людського сприйняття. Підсумкове поле балансу користувача ніде не з'являється, ця величина є поліморфною.

При створенні наступної транзакції деякий сторонній вузол повинен перевірити реєстр - ланцюжок транзакцій, пов'язаних хешами попередніх транзакцій, щоб переконатися, що кількість входів більша або дорівнює кількості виходів. По суті, це перевірка, щоб підтвердити, чи має відправник достатньо коштів. Враховуючи, що один відправник може мати кілька вхідних і вихідних транзакцій, ми виводимо наступну формулу для визначення достовірності суми

переказу:

, де i = сума всіх вхідних переказів, o = сумою вихідних переказів

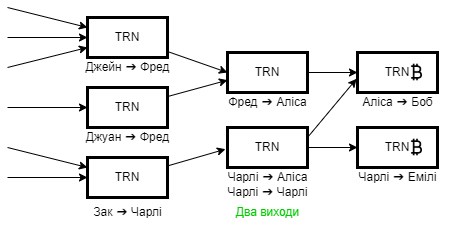


Рисунок 3 Спрощений ланцюжок транзакцій

Блокчейн можна реалізувати різними способами, навіть якщо сфера застосування відома заздалегідь. Наприклад, якщо програма, що розробляється на основі блокчейну, є платіжною системою, то отримання балансу користувача вже може бути реалізовано двома способами: детермінованим і недетермінованим, не кажучи вже про склад блоку, обмеження блоку, винагороду за майнінг тощо. Така ситуація є більш негативною, оскільки безпека кінцевого продукту не буде визначатися загальноприйнятими стандартами, які пройшли відкритий і тривалий аналіз. Щоб протистояти цьому фактору, поширеною практикою є дотримання стандартів де-факто, таких як біткоїн (який діє як класична платіжна система) та Ethereum (який діє як платформа для смарт-контрактів).

### **Транзакції та UTXO модель**

Модель UTXO (Unspent Transaction Output) - це підхід до управління транзакціями без додаткової необхідності підтвердження права власності на кошти. Це означає, що при створенні переказу ми можемо розпоряджатися тільки всією частиною отриманих монет, пам'ятаючи про функціонал балансу. Оскільки фізичного поняття балансу не існує, це просто функція, яка рекурсивно відновлює кількість зобов'язань перед об'єктом по ланцюжку транзакцій. Таким чином, ви можете витратити залишок тільки повністю. Часткова витрата призведе до розгалуження гілок і неузгодженості в ланцюжку блокчейну. Механізм переказу монет працює наступним чином: об'єкт-відправник повинен переказати необхідну суму об'єкту-одержувачу, а якщо сума менша за залишок, то об'єкт повинен повернути залишок собі (див. рис. 2.1).

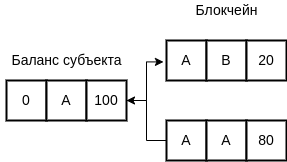
Вирішення проблеми подвійних витрат є дуже важливим. Враховуючи, що в мережі немає посередника або валідатора, якому за замовчуванням довіряють всі без винятку вузли, виникає проблема з прив'язкою транзакцій. Відправник A може відправити користувачеві B транзакцію, еквівалентну його балансу, а потім відправити таку ж суму користувачеві C. Мережа повинна мати певний алгоритм і ряд властивостей, щоб запобігти подібним типам атак. Для цього кожна транзакція містить посилання на попередню, тобто останню загальноприйняту на даний момент, і власний унікальний ідентифікаційний номер, який буде використовуватися для початку нових транзакцій в майбутньому. У Bitcoin такий ідентифікатор генерується за допомогою хешу самої транзакції і транзакції, формуючи таким чином спрямований список і захищаючи транзакцію від майбутніх змін.

Рисунок 4 Специфікація UTXO транзакції

Після цього транзакція потрапляє в пул пам'яті - місце, де вона вже потрапила в мережу, але ще не була записана в жодному з блоків, а отже, не була ефективно задекларована. Транзакція залишається в цьому стані до того моменту, поки її не буде додано до блоку. Блок - це колекція транзакцій розміром в один мегабайт. Блоки необхідні для оптимізації продуктивності мережі, оскільки підтвердження тисячі транзакцій одночасно є менш ресурсоємним і більш ефективним. Формування блоку починається зі створення реєстру - криптографічно підтвердженого впорядкування транзакцій, покликаного запобігти вразливості подвійних витрат. Після формування реєстру блок хешується за допомогою алгоритму дерева Меркла.

### **Конфіденційність**

На відміну від традиційних банківських систем, концепція конфіденційності в системах блокчейн принципово відрізняється. У банківській моделі сам банк є довіреною особою, яка захищає нашу конфіденційну інформацію, маючи при цьому повний доступ до неї. На противагу цьому, сучасні криптовалютні протоколи працюють за зовсім іншим принципом приховування. Вся інформація про рахунки, транзакції та суми на них є загальнодоступною для кожного учасника мережі. Більше того, кожен учасник зобов'язаний зберігати цю інформацію самостійно.

Незважаючи на таку прозорість, пов'язати номер рахунку з його власником досить складно. Це насамперед пов'язано з псевдонімічною природою блокчейн-транзакцій, коли користувачів ідентифікують за допомогою їхніх відкритих ключів, а не за їхніми реальними іменами та прізвищами. Однак такий підхід також створює низку унікальних викликів, які потенційно можуть порушити цю концепцію конфіденційності.

Однією з таких проблем є можливість "аналізу ланцюжка", коли транзакції можуть бути відстежені і потенційно пов'язані з користувачами. Щоб вирішити цю проблему, деякі користувачі вдаються до таких практик, як використання "міксерів" - сервісів, які змішують кошти користувача з коштами інших, щоб приховати слід, що веде до користувача. Інша поширена практика - створення нового гаманця для кожної транзакції, що ускладнює прив'язку транзакцій до одного користувача. Простота створення гаманця, яка полягає у створенні нової пари ключів, сприяє такому підходу.

Однак важливо зазначити, що хоча ці методи можуть підвищити рівень конфіденційності, вони не є надійними, і їм притаманні власні ризики та виклики. У криптовалютній спільноті тривають дебати про те, як найкраще збалансувати потребу в конфіденційності з потребою в прозорості та безпеці.

### **Асиметричне шифрування**

Для того, щоб зрозуміти реалізацію мережі в потенційно ворожому середовищі, важливо бути знайомим з основами асиметричного шифрування - ключовою особливістю технології блокчейн. Ми не будемо заглиблюватися в математичні аспекти, такі як генерація випадкових байт для пари ключів або односторонні функції, що використовуються в алгоритмах шифрування, а зосередимося на протоколі, який лежить в основі шифрування і логіці цифрового підпису.

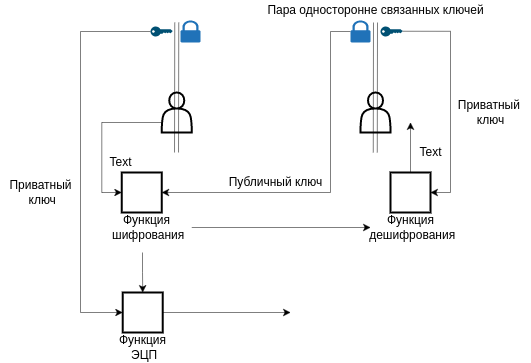


Рисунок 5 Специфікація взаємодії двох суб'єктів у явно ворожому середовищі

Як показано на Рисунку 2.3, для шифрування даних нам потрібні два однонаправлено пов'язані ключі, відомі як "пара ключів". Ця пара складається з приватного ключа, який тримається в секреті, і публічного ключа, який транслюється відкрито. Зв'язок між цими двома ключами такий, що, маючи приватний ключ, ми можемо легко обчислити відповідний публічний ключ, але зворотна операція - обчислення приватного ключа з публічного ключа - є обчислювально нездійсненною.

Саме ця властивість забезпечує безпечну комунікацію в потенційно ворожому середовищі. Повідомлення, зашифроване відкритим ключем одержувача, може розшифрувати лише власник відповідного закритого ключа. Навіть якщо відкритий ключ є загальнодоступним і процес шифрування відомий, безпека шифрування полягає в неможливості обчислення закритого ключа з відкритого.

Той самий принцип використовується для створення цифрового підпису. Відправник підписує повідомлення своїм закритим ключем, а будь-хто, хто має відкритий ключ відправника, може перевірити підпис, тим самим підтверджуючи, що повідомлення дійсно було відправлено відправником і не було підроблено в дорозі. Цей механізм допомагає запобігти атакам типу "людина посередині" (Man-in-the-Middle, MITM), коли зловмисник перехоплює і потенційно змінює повідомлення між двома сторонами.

Таким чином, поєднання асиметричного шифрування та цифрових підписів забезпечує надійний механізм для безпечної комунікації та перевірки транзакцій в мережі блокчейн.

### **Proof-Of-Work**

У цьому розділі обговорюється реалізація консенсусу в системі блокчейн Біткоїн. Сатоші Накамото у своєму документі про Біткоїн посилався на систему Hashcash Адама Бека, яка вперше представила алгоритм Proof of Work (PoW) як універсальну технологію захисту від спаму.

Концепція, що лежить в основі, проста: Якщо вузол повинен виконати певну обчислювальну роботу, перш ніж підтвердити блок, йому буде невигідно атакувати мережу тисячами транзакцій в секунду. Хоча ця технологія може використовуватися в інших системах для запобігання спаму, в контексті Біткоїна консенсус відіграє життєво важливу роль у рівномірному розподілі емісії монет і виборі лідера для додавання нового блоку до блокчейну.

Біткоїн використовує алгоритм хешування SHA-256. Він бере набір транзакцій у блоці і повертає 256-бітний хеш. Встановивши правило, що мережа прийматиме хеш лише з певною кількістю початкових нулів, можна збільшити складність пошуку відповідного хешу. Відповідно, це зменшує діапазон прийнятних хешів і збільшує час, необхідний для хешування.

Щоб гарантувати, що однакові вхідні дані не завжди дають однаковий хеш, було введено поняття, яке називається "nonce". Подаючи випадкові дані (nonce) разом з транзакціями на функцію хешування, можна згенерувати різні хеші для одного і того ж блоку. Вузли будуть продовжувати хешування з різними nonce до тих пір, поки один з них не знайде хеш в прийнятному діапазоні.

Обчислювальна потужність мережі, яка залежить від кількості вузлів-учасників, є динамічною. Якщо кількість майнерів подвоюється, швидкість підтвердження блоків також подвоюється, що прискорює процес підтвердження, але потенційно призводить до збільшення навантаження на мережу і швидкості емісії монет. Щоб запобігти цьому, мережа коригує рівень складності приблизно кожні 2016 блоків, або приблизно кожні два тижні, щоб підтримувати середній час підтвердження блоку на рівні 10 хвилин.

Форки, або розгалуження в блокчейні, можуть виникати, коли два різних блоки одночасно знаходять правильний хеш. У цій ситуації мережа фактично розділяється на дві частини, кожна з яких продовжує свою гілку. Ця проблема вирішується дотриманням правила "найдовший ланцюжок перемагає": та гілка, яка першою додає наступний блок, визнається головною гілкою. Вузли, які працювали на коротшій гілці, повинні перейти на нову головну гілку. Тому для забезпечення підтвердження транзакції важливо дочекатися більш ніж одного підтвердження від мережі.

### **Proof-Of-Elasped-Time**

Proof of Elapsed Time (PoET) - це алгоритм консенсусу, який використовується в системах блокчейн, зокрема на платформі Intel Sawtooth Lake. PoET розроблений для забезпечення справедливого і високомасштабованого процесу підтримки консенсусу в децентралізованій мережі, при цьому пом'якшуючи деякі з істотних проблем споживання ресурсів, пов'язаних з іншими алгоритмами консенсусу.

Основа PoET відносно проста. Мета полягає в тому, щоб визначити легітимність і порядок транзакцій в децентралізованій системі, що є критично важливим для будь-якої мережі блокчейн. Замість того, щоб покладатися на величезні обчислювальні потужності, як в Proof of Work (PoW), або володіння великою часткою в мережі, як в Proof of Stake (PoS), PoET використовує систему випадкової лотереї для вибору вузла, який додає наступний блок до ланцюжка.

В алгоритмі PoET кожен вузол, що бере участь в мережі, генерує випадковий час очікування і засинає на цей час. Вузол, який прокидається першим - тобто вузол з найкоротшим часом очікування - додає новий блок до блокчейну і транслює його решті мережі. Цей процес повторюється для додавання кожного нового блоку. Це наче кожен вузол - поет, який чекає на натхнення; той, хто прокинеться першим, напише наступний рядок "поеми", якою є блокчейн.

Критично важливим аспектом PoET є забезпечення цілісності часу очікування. Для цього PoET використовує розширення Intel Software Guard Extensions (SGX), які дозволяють програмам запускати надійний код у захищених контейнерах, відомих як анклави. SGX гарантує, що код, який генерує випадковий час очікування і спить протягом цього часу, працює, як очікувалося, і не був підроблений, тим самим забезпечуючи чесність лотерейної системи.

PoET має кілька переваг як алгоритм консенсусу. Він є енергоефективним, оскільки вузлам не потрібно виконувати обчислювально інтенсивні завдання, і вони можуть переходити в режим сну з низьким енергоспоживанням під час очікування. PoET також підтримує високий ступінь масштабованості, оскільки додавання нових вузлів до мережі не призводить до значного збільшення обчислювальної потужності, необхідної для досягнення консенсусу. Нарешті, PoET сприяє справедливості, оскільки кожен вузол, незалежно від його обчислювальної потужності або частки в мережі, має рівні шанси бути обраним для додавання наступного блоку.

Незважаючи на ці переваги, з PoET пов'язані також проблеми і критика. Він покладається на надійне середовище виконання, надане Intel SGX, що викликає занепокоєння щодо централізації та довіри. Крім того, він може стати вразливим, якщо зловмисник знайде спосіб скомпрометувати SGX або маніпулювати процесом генерації випадкових чисел.

На закінчення, Proof of Elapsed Time представляє унікальний та інноваційний підхід до питання консенсусу в мережах блокчейн. Поєднуючи елементи випадковості, справедливості та енергоефективності, він означає значний відхід від традиційних механізмів консенсусу, що вимагають значних ресурсів. Однак, як і всі технології, вона не позбавлена потенційних проблем і повинна постійно перевірятися, тестуватися і розвиватися, щоб зменшити будь-які вразливості і підтримувати цілісність систем, які вона підтримує.

### **Майнінг**

Майнінг нерозривно пов'язаний з механізмом консенсусу щодо доказів роботи. Через відсутність центрального органу влади в мережі за замовчуванням виникає проблема, відома як "проблема візантійських генералів". Це класичний виклик в криптографії, що передбачає прийняття рішень в потенційно ворожому середовищі. У Біткоін вона вирішується шляхом випадкового вибору вузла, блок якого визнається дійсним всією мережею.

По суті, учасники мережі, включаючи майнерів, грають у своєрідну криптографічну лотерею. Вони перевіряють цілісність блоку, порівнюючи його хеш з хешами інших вузлів; якщо в блоці є навіть незначні зміни, хеш буде кардинально відрізнятися, що робить подальші зусилля з майнінгу марними, оскільки мережа відхилить такий блок.

Після перевірки блоку всі учасники намагаються знайти відповідний хеш за допомогою nonce. Як тільки такий хеш знайдено, успішний вузол отримує винагороду від бази монет і право додавати та розповсюджувати новий блок. Враховуючи, що пошук відповідного хешу є обчислювально інтенсивним завданням, майнеру, як правило, невигідно намагатися обдурити мережу; будь-які спроби змінити дані блоку, швидше за все, будуть помічені іншими вузлами, які потім відхилять блок майнера.

База монет - це спеціальна сутність в блокчейні, яка не має приватного ключа. Вона служить для виплати майнерам з власних резервів за їх внесок в мережу. База монет може бути відновлюваною або невідновлюваною.

У випадку з невідновлюваною базою монет

1. Майнер видобуває блок
2. Монетна база платить майнеру зі своїх резервів
3. Майнер також збирає частину комісії за транзакції

У випадку з відновлюваними монетними базами:

1. Майнер видобуває блок
2. Частина комісії за транзакції перераховується на монетну базу
3. Коін-чейн платить майнеру зі своїх резервів

Коли блок видобувається декількома майнерами одночасно, виникає конкуренція або "майнінгова гонка". Майнери з більшою обчислювальною потужністю мають більше шансів на перемогу. Однак координація цих зусиль для уникнення дублювання є складним завданням. Щоб вирішити цю проблему, створюються майнінг-пули, які об'єднують майнерів для роботи над однією проблемою і спільного використання ресурсів. Ці пули координують зусилля окремих майнерів, щоб вони не дублювали роботу один одного.

**РОЗДІЛ 3**

# Форування блоків

Почати розгляд блоку варто з самої мінімальної його версії, де він здатний залишатися і функціонувати як об'єкт в блокчейне. Так, для блоку досить тільки одного поля - хешу попереднього блоку. Ця урізана версія проста для розуміння, але також породжує теоретичну базу. Так, наприклад, давайте уявимо, що перед нами є певний блок В0, який представляє початок ланцюжка, в термінології це блок генезису. Суть його в тому, що він не має попереднього блоку і є єдиним прабатьком всіх наступних. Ви можете думати про блок B0 наступним чином:

B0 = <RANDSTR>

Випадковий рядок в блоці генезису замінює хеш блоку і в теорії він може бути будь яким, навіть не випадковим, так як це ніяк не впливає на безпеку. Всі наступні блоки будуть генеруватися на основі наступної формули:

Bi = hash(Bi-1),

де hash()- криптографічна хеш-функція[[1]](#footnote-1)

Таким чином, виходить ланцюжок хешів, яку тільки із застереженнями можна назвати блоком. Щоб надати блоку «сенс», потрібно помістити дані (D) в блок для зберігання, але таким чином,

щоб ці дані були зафіксовані в результаті хешування. Тепер блок ускладнився і замість одного значення (хешу) став зберігати два (дані і хеш), тим самим ставши масивом:

B0 = [<NULL>, <випадковий рядок>]

Bi  = [D, hash(D || Bi-1{2})]

У цьому прикладі була введена запис «||», що означає об'єднання рядків і запис «Bі-1{2}», що означає взяти другий елемент з масиву Bі-1 (індексація для масиву почнеться з одиниці).

Цей запис вже більше схожа на блокчейн, адже в ній зберігаються хоч якісь значення, нібито корисні користувачеві. Але може виникнути логічне питання: навіщо все це було зроблено? І відповідь - за безпеку (хоча на даному етапі проблем ще дуже багато). Суть полягає в тому, що більше неможливо змінити значення Bi{1} (або Bi-1{2}), не впливаючи на Bi{2}, інакше зміна буде виявлено, оскільки значення блоку не буде відповідати його хешу.

Ситуація для зловмисника буде плачевною в той момент, коли йому потрібно буде змінити значення блоку в середині або на початку ланцюжка, так як йому доведеться міняти всі наступні хеші, які надходять після зміни блоку. Але проблеми в цій схемі все ж залишаються.

Що робити, якщо грошові перекази зберігаються в блокчейні? Буде дуже прикро, якщо зловмиснику вдасться змінити ваше значення, що зберігається в блоці, на значення набагато більше, або згенерує новий блок, в якому він вказує вас як відправника.

Щоб вирішити цю проблему, необхідно ознайомитися з таким терміном, як цифровий підпис[[2]](#footnote-2).

Цифровий підпис допомагає ідентифікувати творця блоку, тим самим усуваючи моменти, коли хтось може видати себе за іншу людину. Саме поняття цифрового підпису несе в собі ще два терміни - address (відкритий ключ) і wallet (закритий ключ).

Підпис слід наносити на хеш блоку, а не на його значення за двома ознаками. По-перше, хеш завжди має фіксовану довжину, відповідно, і швидкість підписання не буде змінюватися. По-друге, підпис також вплине на хеш-інформацію попереднього блоку, що повністю зв'яже підпис з попередніми блоками.

Блок ускладнився трьома елементами (значення, хеш, підпис):

B0 = [<NULL>, <RANDSTR>,<NULL> ]

Bi = [D, hash(D || Bi-1{2}), sign(Bi{2})]

Тепер, якщо зловмисник хоче змінити значення Bi{1}, йому потрібно буде змінити значення Bi{2}, але якщо він змінить значення Bi{2} він не зможе змінити значення Bi{3} так що цей блок продовжує ідентифікуватися як до зміни. Або іншими словами, посередник зможе змінити блок, але тільки в тому випадку, якщо він поставить свій підпис на блоці і аналогічно змінить всі наступні блоки. Таким чином, зловмиснику доведеться міняти весь блокчейн, що звичайно є досить складним процесом, але можливим.

Навіть якщо ваш блок знаходиться на початку ланцюжка, а ланцюжок - мільйон блоків, сучасний комп'ютер зможе змінити значення в ланцюжку і потім генерувати всі наступні хеші з підписами зловмисника, тим самим стираючи всі реальні блоки. Генерація всіх блоків, які в даний момент існують в блокчейні біткоіни, займе не більше 10 хвилин на сучасному процесорі.

У цьому випадку на допомогу приходить доказ роботи[[3]](#footnote-3). Це ще більше ускладнення блоку, до якого додається такий термін, як складність розрахунку. Ця складність заснована на математично складній задачі, подобі знаходження потрібного хешу. Звичайний комп'ютер може витрачати на такі завдання кілька годин, днів, а то й місяців і років, не кажучи вже про вміння людини знайти рішення своїми руками. Для регулювання складності була придумана задача обчислення не всього хешу, а тільки його частини. Припустимо, ви проходите через число і хешуєте його, вам потрібно знайти значення, що швидко псується, щоб воно починалося з чотирьох нульових байтів. Розмір хешу, а точніше кількість нульових байтів на початку називається ціллю. Якщо такий хеш знайдений, то проблема вирішена. Але тут однозначно є мінус, якщо хоча б одна людина знайде хешоване число, яке явно почнеться з великої кількості нулів, то доведеться змінити спосіб обчислення складної задачі, наприклад, перейти на одиниці, потім на двійки, трійки, четвірки і так далі. Це явно незручно, відповідно, був придуманий вдосконалений спосіб виконання цього завдання, що робити, якщо хешувати не просто збільшене число, а хешувати об'єднання збільшеного числа з поточним хешем блоку? Завжди буде отриманий випадковий результат і отримане інкрементне число не можна буде використовувати знову на іншому блоці. Пошук потрібного хешу ще називають Майнінг.

Блок тепер буде виглядати так:

B0 = [<NULL>, <RANDSTR>,<NULL> ,<NULL> ]

Bi = [D, hash(D || Bi-1{2}), sign(Bi{2}), pow(Bi{2})],

, де pow повертає оказію x = Bi{4} в якому хеш(x || Bi{2}) <= TARGET.

Слід зазначити, що функція військовополонених призначена не тільки для підтвердження роботи, що сповільнило б генерацію блоків і розтягнуло б емісію монет в мережі. В основі мети доказу роботи лежить рішення двох математичних задач. Перший - це протокол відбору лідера, де лідер, а саме майнер, буде обраний гідним внести якусь інформацію в мережу. Другий - розподілене голосування, де вага голосу визначається комп'ютером учасника. У цьому випадку результат військовополоненого, число, яке доводить, що

1.Майнер використовував правильну ціль

2.Поточна складність майнінгу блоків

Також ряд неоднозначних питань виникає при об'єднанні учасників в пули і зіставленні майнера зі знайденим хешем. Тобто блок не містить ніякої інформації про те, хто замінував. Уявімо собі таку ситуацію, що ви самі не видобуваєте блок, а перекладаєте це завдання на групу інших людей, потужність комп'ютера яких в кілька разів перевищує ваш комп'ютер і за це ви платите певний відсоток від перерахованих коштів. Тоді одна людина з цієї групи нарешті знаходить нонс і кладе його в блок. Але в цій групі є нападник, він копіює нонс і пересилає його вам. Тепер ви отримали дві відповіді про знайдений хеш, кому довіряти? Можна сказати, хто приніс першим, той отримає винагороду, але можна зробити це і іншим, більш правильним способом. Нехай майнер сам створить блок, а ви відправите йому тільки транзакцію (а саме перші три поля блоку), потім майнер введе в блок свій хеш і підпис, а на своєму хеші виконає операцію військовополонених.

Блок ще більше ускладнюється:

B0 = [<NULL>,<NULL> , <RANDSTR>,<NULL> ,<NULL> ,<NULL> ]

Bi = [[D, hash(D), sign(Bi{1}{2})], M, hash(M || Bi{1}{2}|| Bi-1{3}), signM(Bi{3}), pow(Bi{3})]

Коли майнер знайде оказію, число x = pow(Bi{3}), ніхто інший не зможе скопіювати це число і підставити його для свого результату, так як отримане значення не буде підходити під хеш іншого блоку, де буде вказано інший майнер. Але навіть незважаючи на це, необхідна транзакція надходить у блокчейн, зі значенням, хешем значення та хеш-підписом. Тобто все як годиться. Але і на цьому етапі є лазівка. Ви можете замінити знак(Bi{1}{2}) без наслідків і ніхто цього не помітить. Звичайно, питання може бути різним, для яких цілей існуючий підпис сторонньої особи захоче змінити на його підпис, при цьому підлаштовуючись під його хеш? Але на жаль, статися може все, що завгодно і тому його теж слід вважати вразливістю. У виправленні цієї вразливості може допомогти розкриття нутрощів магічного значення D.

І знову блок починає ускладнюватися:

B0 = [<NULL>, <NULL>, <RANDSTR>,<NULL> ,<NULL> , <NULL>]

Bi = [[[S, R, V], hash(Bi{1}{1}), signS(Bi{1}{2})], M, hash(M || Bi{1}{2} || Bi1{3}), signM(Bi{3}), pow(Bi{3})]

При цьому S - відправник, R - приймач, V - значення (саме значення або кількість умовних відправлених одиниць). Також змінено вигляд знака(Bi{1}{2}) на signS(Bi{1}{2}), вказавши, чий це підпис. Тепер підпис не можна замінити, не зачепивши хеш. Також не можна змінювати творця підпису, адже потрібно буде змінити хеш-значення, що неминуче призводить знову до заміни підпису. Підпис майнера також проблематично змінити, адже його ім'я вказано в хеші. Доказ роботи також не вкрадуть, адже він прив'язаний до хешу. Здається, ніби ось вона, ідеальний захист! Але на жаль, немає, вразливості все ж є.

Пам'ятайте, що нікому не можна довіряти блокчейну? Власне, це і стосується шахтарів. Але як майнер може завдати шкоди? Адже була зроблена хеш-підпис, а хеш вказує на відправника, одержувача і значення, тобто нічого з цього змінити не можна, і якщо змінити його повністю, що аналогічно простому ігноруванню додавання транзакції в блок. Це все вірно, але є одна деталь, яка дуже дратує - угода ні до чого не прив'язана. Відповідно, майнер або хтось інший зможе скопіювати всю транзакцію і продублювати її знову, при цьому вона також буде дійсна, адже всі хеші і підписи правильні (адже вони дійсно реальні). Щоб позбутися від цієї вразливості, можна просто вказати хеш попереднього блоку в угоді.

Блок трохи ускладнився:

B0 = [<NULL>,<NULL> , <випадковий рядок>,<NULL> ,<NULL> ,<NULL>]

Bi = [[[S, R, V], hash(Bi{1}{1} || Bі-1{3}), signS(Bi{1}{2})], M, hash(M || Bі{1}{2} || Bі-1{3}), signМ(Bі {3}), (powBi {3})]

Тепер, навіть якщо майнер захоче продублювати транзакцію, вона буде перебувати під новим блоком, і хеш попереднього блоку більше не буде збігатися з хешем, який є в транзакції. І якщо майнер спробує залишити змінені значення, навіть незважаючи на невідповідність, то інші майнери просто відкинуть його блок як недійсний.

А тепер у нас блоки вишикувалися в ланцюжок, всі вони захищені від підміни, зміни, дублювання і одночасно від всіх користувачів. Але що робити, якщо майнерів кілька і у кожного з майнерів різний ланцюжок блоків? Як вони домовляться про загальний блокчейн? Pow - це різновид комп'ютерного голосування. Уявіть, що два або більше майнерів знайшли відповідний блок одночасно або майже одночасно, і поширили обидва блоки на мережу. У цей момент блокчейн розпаралелюється на кілька гілок, і перетворюється в дерево. Таким чином, UTXO перестає працювати і стає не зовсім зрозуміло, як перевіряти транзакції в такому деревному ланцюжку. Проблема вирішується за допомогою Майнінг. Зараз змагаються група вузлів, які взяли блок першого майнера, і група, яка взяла блок другого. Той, хто першим знайде блок, має право замінити і відкотити ланцюг протиборчої групи вузлів. Отже, чим більше комп'ютерна група, тим більше шансів, що вона першою знайде хеш. Тому, якщо більшість мережі продовжувала видобувати будь-який з ланцюгів, вони голосували за цю гілку блокчейну. Такі суперечливі ситуації називаються софт-форк, коли блокчейн розділений на кілька гілок, але все ж є можливість вибрати тільки одну «правильну» ланцюжок.

А крім софт-форків, існують ще й так звані хард-форки. Тверді вилки, на відміну від м'яких, не з'являються самі по собі. Вони використовуються, коли необхідно змінити або поліпшити додаток, або створити новий блокчейн на основі існуючого. Це призводить до того, що нова гілка не буде «зливатися» зі старою, тим самим стаючи несумісною.

Також кожен блок повинен випускати монети в мережу. Таким чином, знаючи все це, необхідно змінити блок генезису:

B0 = [[[S, M, V], <NULL>, <NULL>], <NULL>, hash(B0{1}{1}), <NULL>, <NULL>,<NULL> ],

де S - сховище, M - добувач блоку генезису, V - винагорода блоку генезису. Як тільки блок генезису генерується, починає діяти сховище, яке має певний і обмежений запас ресурсів.

Начебто все вже сказано, але в блокчейні все ще є ряд тонкощів.

По-перше, якщо транзакція відправляється різним майнерам одночасно, то у майнерів є конкуренція або гонка на видобуток. Шанси на перемогу більші у того, хто має більше влади, але питання в іншому, як координуються шахтарі? Адже в теорії майнери можуть добувати одні й ті ж значення, тим самим витрачаючи час і сили. Для такого випадку створюється пул серверів, до якого підключається група майнерів, зацікавлених у вирішенні одного завдання і розподілі ресурсів між собою. Сервери пулу координують дії кожного окремого майнера, щоб вони не перетиналися в пошуку одних і тих же значень.

По-друге, транзакцій в блоці може бути кілька (і це найчастіше буває). В цьому випадку виникає ще одна вразливість, а саме дублювання угоди в одному блоці. Щоб цього не сталося, потрібно вставляти в кожну транзакцію випадкове число (r) і хешувати його разом з усією іншою інформацією.

По-третє, ви можете використовувати позначки часу (Time), які вкажуть час створення блоку, для його подальшого відстеження в історії та визначення поточної ємності блокчейну майнерами[[4]](#footnote-4). Все це можливо тільки за умови грамотної синхронізації з сервером, який представляє час.

В результаті остання версія блоку буде виглядати так:

B0 = [[[[, S, M, V],<NULL> ,<NULL> ]],<NULL> ,<NULL> , hash(B0{1}{1}{1}),<NULL> ,<NULL> ,<NULL> ]

Bi = [[[[r, S, R, V], hash(Bi{1}{1}{1} || Bi-1{4}), signS(Bi{1}{1}{2})], [...], ... ], M, Time, hash(M || Time || (Bi{1}{1}{2} || Bi{1}{...}{2}) || Bi-1{4}), signM(Bi{4}), pow(Bi{4})]

**РОЗДІЛ 4**

# Топологія мережі, Gossip протокол та її складові

Чим Якщо заглибитися у вивчення мережі Біткоїн, то вона може здатися менш децентралізованою, ніж здається на перший погляд. Ця мережа складається з серверів часу, пул-серверів, відповідальних за координацію дій вузлів, і самих вузлів, що діють як розподілений сервер, а користувачі, які здійснюють транзакції, є клієнтами. Метою цього розділу є вивчення особливостей блокчейну з точки зору мережі та встановлення зв'язків між її учасниками. Зокрема, Біткоїн можна розглядати як оверлейну однорангову мережу. Оверлейна мережа - це загальний випадок логічної мережі, створеної поверх іншої мережі. Вузли в оверлейній мережі можуть бути з'єднані або через фізичне з'єднання, або через логічне з'єднання, з одним або декількома відповідними маршрутами від фізичних з'єднань в базовій мережі. Однорангова мережа - це комп'ютерна мережа, заснована на рівності її учасників. Часто в такій мережі немає виділених серверів, і кожен вузол (пір) виступає одночасно і клієнтом, і сервером. Така організація дозволяє підтримувати працездатність мережі при будь-якій кількості та комбінації доступних вузлів. Всі вузли є учасниками мережі.

Існує два основних типи мереж: клієнт-серверні (багатосторонні) та однорангові. Клієнт-серверні мережі припускають існування двох сутностей - клієнта і сервера, де клієнт може робити запити, а сервер може відповідати. Однорангові мережі передбачають рівність учасників у діях, тобто користувач у такій мережі є одночасно і клієнтом, і сервером (за термінологією клієнт-серверних мереж). Такі користувачі називаються вузлами. Таким чином, однорангові мережі є підмножиною децентралізованих мереж, які, в свою чергу, є частиною множини розподілених мереж.

Розподілені мережі охоплюють як однорангові мережі з топологією "всі до всіх", так і децентралізовані мережі. Ми не будемо зосереджуватися на інших типах розподілених мереж в цьому обговоренні.

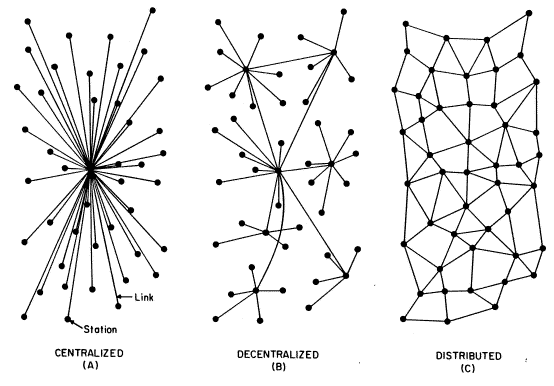


Рисунок 6 Графічне представлення комп'ютерних мереж

На додаток до раніше розглянутих клієнт-серверних та однорангових мереж, існує третій тип мереж, який називається гібридною мережею. Відмінною рисою гібридної мережі є наявність як клієнтських, так і серверних об'єктів, як у клієнт-серверній архітектурі, але з певним нюансом: сервер сам по собі є одноранговою мережею, або, іншими словами, мережею вузлів. На графічному зображенні вище мережа A представляє клієнт-серверну мережу, B символізує гібридну мережу, а C позначає однорангову мережу.

Однак така класифікація гібридної мережі може викликати кілька запитань. Наприклад, якщо фізична особа запускає веб-сайт з розподіленим між серверами сховищем даних, чи можна вважати таку мережу гібридною мережею? Більше того, чи можна вважати мережу гібридною, якщо вона складається лише з одного вузла?

Щоб відповісти на ці питання, нам потрібно уточнити наше розуміння гібридної мережі. У контексті гібридної мережі одноранговий компонент, як правило, відноситься до декількох взаємопов'язаних вузлів, які поділяють обов'язки та ресурси. Таким чином, веб-сайт з розподіленим зберіганням даних між серверами можна вважати гібридною мережею, якщо ці сервери взаємодіють і працюють в одноранговому режимі.

З іншого боку, мережа з одним вузлом зазвичай не вважається гібридною мережею. Це пов'язано з тим, що ключовою характеристикою гібридної мережі є поєднання клієнт-серверної та однорангової архітектур. Одиночний вузол не буде демонструвати одноранговий аспект цього визначення, оскільки немає інших вузлів, з якими він міг би взаємодіяти. По суті, гібридна мережа - це інтеграція моделей клієнт-сервер і однорангових мереж, де серверний компонент сам по собі є одноранговою мережею. Хоча це визначення може не охоплювати всі потенційні варіації, воно пропонує практичну основу для класифікації та розуміння різних мережевих архітектур, особливо в контексті блокчейну та аналогічних децентралізованих систем.

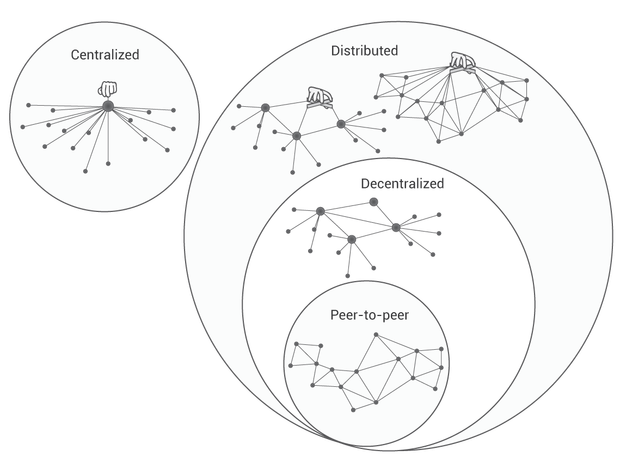


Рисунок 7 Графічне представлення ієрархії комп'ютерних мереж

Щоб продовжити обговорення мережевих топологій, важливо визначити фундаментальну відмінність між багатопартійними та гібридними мережами. Насправді, така різниця існує. Вузли в гібридних мережах, на відміну від серверів у багатосторонніх мережах, працюють спільно, але по суті не довіряють один одному. Ця характеристика передбачає децентралізований характер, оскільки вузли не перебувають під контролем одного суб'єкта або невеликої групи людей. Однак вона також передбачає, що якщо є лише один вузол або якщо всі вузли перебувають під контролем однієї групи, мережа втрачає свої гібридні властивості і перетворюється на багатопартійну мережу.

Виходячи з цього розуміння, ми можемо окреслити наступні зв'язки між учасниками мережі блокчейн:

1. Клієнт -> [однорангова мережа] -> Вузол.
2. Вузол -> [однорангова мережа] -> Вузол.
3. Вузол -> [одноранговий] -> Сервер пулу.
4. Вузол -> [одноранговий] -> Сервер часу.

* Клієнти взаємодіють з вузлами для отримання балансу, інформації про блок або для запису транзакції всередині блоку.
* Ноди взаємодіють з іншими нодами для підтримки узгодженої версії блокчейну. Вузол може надсилати запити до іншого вузла на додавання нового блоку до блокчейну.
* Вузли запитують вказаний діапазон майнінгу у сервера пулу.
* Ноди запитують поточний часовий статус у сервера часу.

Розуміння цих взаємодій покращує наше розуміння того, як учасники мережі блокчейн взаємодіють і співпрацюють, незважаючи на притаманний такій децентралізованій системі брак довіри. Це підкреслює тонкий баланс між співпрацею і конкуренцією, який лежить в основі безпеки і надійності мережі.

Хоча багатосторонні зв'язки є більш численними, однорангові зв'язки мають першорядне значення. Одне з ключових питань полягає в тому, чи погіршується відмовостійкість мережі блокчейн при впровадженні однорангового зв'язку. Наприклад, розглянемо сценарій, коли зловмисник намагається відключити сервер пулу від мережі за допомогою DDoS-атаки. У разі успіху деякі вузли втрачають доступ до сервера. Ці вузли можуть або підключитися до іншого сервера пулу, або самостійно генерувати блоки на основі випадкового числа. У будь-якому випадку, мережа продовжує функціонувати, хоча потенційно з погіршеною продуктивністю.

Розглянемо сервер часу, який відрізняється від пул-сервера тим, що використовується лише один раз перед запуском вузла. По суті, він запитує точний час у сервера, встановлює цей час на локальній машині, а потім запускає вузол. Атаки на сервер впливатимуть лише на нові вузли, в той час як існуючі вузли продовжуватимуть працювати. Враховуючи, що зазвичай існує декілька серверів часу, успішне виконання DDoS-атаки здається складним завданням. Навіть якщо всі сервери часу будуть відключені від мережі, блокчейн продовжить функціонувати, а поточний час буде підтримуватися вузлами. Отже, новий вузол може запитувати час не напряму у серверів часу, а опосередковано, через існуючий вузол блокчейну. Таким чином, відмовостійкість блокчейну значною мірою залежить від однорангової архітектури, а відключення від серверів потенційно може знизити продуктивність певної групи майнерів, але не зменшити рівень відмовостійкості.

З огляду на притаманну мережі блокчейн децентралізацію, виникає слушне запитання: як клієнти (і вузли) підключаються до інших вузлів? Звідки вони отримують початковий список необхідних адрес? Це питання є поширеним в контексті децентралізованих мереж, причому різні мережі вирішують його по-різному. Найпоширенішим рішенням є використання сторонніх каналів зв'язку. Наприклад, може бути створено сервер, на якому зберігається список активних вузлів. Вихід з ладу такого сервера не зруйнує мережу, але ускладнить доступ до інформації про неї та її зростання (суттєва проблема лише на початковому етапі формування блокчейну). Цей список будуть розширювати самі вузли (тобто вводити свої адреси на сервері), оскільки вони мотивовані потенційним прибутком від майнінгу. Чим частіше адреса майнера з'являтиметься на різних серверах, тим частіше клієнти будуть звертатися до нього за допомогою у майнінгу (а самі майнери - за додаванням нових блоків).Якщо в блокчейні враховувати додавання адрес, то кількість зв'язків у мережі збільшиться на два рядки:

1. Клієнт -> [багаторангова] -> Сервер адрес

2. Вузол -> [багаторангова] -> Сервер адрес

Крім того, важливо зазначити, що майнер може надати список всіх інших майнерів, до яких він підключений, тому сервери зазвичай не потрібні. Однак може виникнути питання: через конкурентну природу майнінгу, чи може майнер поділитися скороченим списком адрес, щоб збільшити свої шанси на успішний майнінг? Хоча це можливо, важливо розуміти, що якщо майнер не встигає вчасно розкрити блок і блокчейн оновлюється іншим майнером, блок, видобутий першим, стає недійсним. Згодом майнер має два варіанти: створити власний блокчейн, ігноруючи решту (що є ризикованим, нагадує хардфорк і вимагає залучення майнерів до свого блокчейну), або прийняти оновлений блокчейн і визнати застарілий характер отриманих транзакцій. У цьому випадку клієнти розуміють, що якщо транзакції не включені в новий блок, майнер, ймовірно, не надав повний список активних вузлів. В результаті, клієнтам вигідно підключатися до декількох вузлів і отримувати від них адреси інших майнерів, видаляючи дублікати.

Третій спосіб отримання списку вузлів - через клієнтське програмне забезпечення блокчейну (або блокчейн-вузол). Розробники такого програмного забезпечення можуть за замовчуванням включати в нього список довірених вузлів, які довели свою надійність з плином часу. З оновленням програмного забезпечення цей список також буде оновлюватися.

Щоб підтримувати потужність і доступність мережі блокчейн, всі три методи пошуку адрес повинні використовуватися одночасно. Наприклад, якщо популярність блокчейну зростає, може здатися, що можна відмовитися від першого методу пошуку адреси (за допомогою сервера). Однак це створить вразливість, коли довірені вузли (третій метод пошуку майнерів) вступають у змову і приховують адреси інших вузлів, зменшуючи потужність мережі блокчейн. Це сприятиме "шахрайським майнерам", тоді як інші майнери навіть не матимуть можливості сформувати новий блок на основі транзакцій. І навпаки, якщо прибрати третій спосіб, доступ до сервера стане необхідним на етапі запуску програми. Якщо прибрати перший і другий способи, ймовірність того, що транзакція буде включена в новий блок, знижується (за умови наявності декількох клієнтів з різними списками адрес). Якщо прибрати другий спосіб, сервер стає єдиним відповідальним за надання списку нових адрес. Якщо перший і третій способи вилучити, то мережа блокчейн стає менш зручною для користувача через необхідність ручного налаштування з'єднання.

Обмін даними в біткоїні відбувається за допомогою широкомовлення, зокрема, за допомогою протоколу Gossip. Це набір протоколів, що використовуються в одноранговій комп'ютерній комунікації, в якій інформація поширюється подібно до поширення епідемій. Він передбачає, що кожен вузол або кілька вузлів передають оновлені дані своїм відомим сусідам. Враховуючи, що точна кількість вузлів у мережі Біткоїн невідома, це, ймовірно, найефективніший метод комунікації для вузлів. Існує два раунди обміну:

1. обмін транзакціями

2. обмін блоком

Етап обміну транзакціями відбувається протягом ~10 хвилин, доки не буде знайдена оказія одним із майнерів. Після чого формується блок і відбувається етап обміну цим блоком.

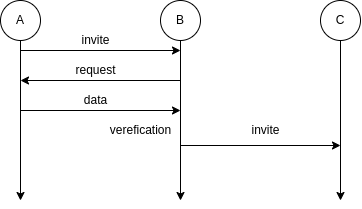


Рисунок 8 Процедура обміну блоком

Алгоритм розподілу блоків у Біткоїні працює в п'ять етапів. На першому етапі вузол A ініціює обмін з вузлом B, який, в свою чергу, обмінюється з вузлом C. Вузол A надсилає свій блок вузлу C, який валідує його і перевіряє його справжність. Потім вузол B також валідує його, перевіряючи на наявність будь-яких змін або структурних невідповідностей, які могли бути зроблені вузлом A. Тільки після цього вузол B ініціює запит на обмін з сусідніми вузлами.

Цей алгоритм має кілька вразливостей, пов'язаних зі швидкістю поширення блоків по мережі. На кожному переході в широкомовній передачі повідомлення піддається затримці поширення. Ця затримка складається з часу передачі та локальної перевірки блоку або транзакції. Час передачі складається з оголошення у вигляді повідомлення-запрошення, запиту від приймаючої сторони і доставки. У той час як повідомлення-запрошення і повідомлення-дані зазвичай невеликі (61Б в більшості випадків, оскільки негайні повідомлення містять тільки один блок або транзакцію, що анонсується), повідомлення-блок може бути значно більшим - до 500кБ на момент написання цієї роботи. Перш ніж блок буде розповсюджений серед сусідів вузла, він проходить верифікацію. Ця верифікація передбачає перевірку кожної транзакції всередині блоку. Перевірка транзакцій, в свою чергу, вимагає випадкового доступу до даних, що зберігаються на дисках.

Нехай ti,j - різниця в часі між першим повідомленням про джерело в мережі і часом, коли вузол j отримує елемент i. Якщо вузол o є джерелом елемента i (тобто, або творцем блоку, або вузлом, який створив транзакцію), то ti,o = 0. Час, в який вузли дізнаються про існування елемента даних, слідує подвійному експоненціальному розподілу. Подібно до випадкового поширення чуток, поширення елемента даних можна розділити на дві фази: початкову фазу експоненціального зростання, на якій більшість вузлів, що отримують повідомлення-запрошення, запитують відповідний елемент даних, оскільки вони його ще не мають, і фазу експоненціального спаду, на якій більшість вузлів, що отримують повідомлення-запрошення, вже мають відповідний елемент даних.

Щоб виміряти затримку поширення, ми реалізували мережевий протокол Bitcoin і підключилися до великої вибірки IP-вузлів. Ми зібрали часову інформацію з блокчейну висотою 180 000 блоків для 10 000 блоків. Інформація про час містить хеш блоку, IP-адресу доповідача та локальну мітку часу, коли було отримано повідомлення. Оцінка для ti,j отримується шляхом віднімання часової мітки першого анонсу блоку від усіх анонсів для цього елемента даних.

На рисунку показано нормалізовану гістограму tb,j для всіх блоків b у вимірюваному інтервалі. Нормалізація дозволяє нам використовувати її як наближену функцію щільності ймовірності швидкості, з якою вузли дізнаються про блок. Примітно, що ми не робимо різниці між розмірами блоків, а натомість агрегуємо за всіма блоками. Медіанний час отримання блоку становить 6,5 секунд, а середній - 12,6 секунд. Довгий хвіст розподілу означає, що навіть після 40 секунд все ще є 5% вузлів, які ще не отримали блок.

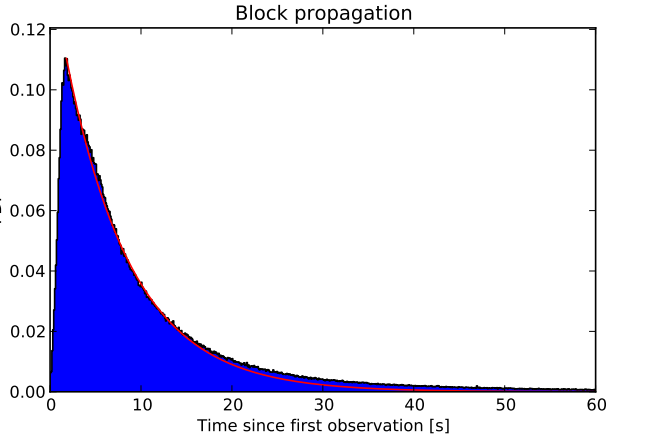


Рисунок 9 Специфікація поширення блоку у мережі біткоїну

Дійсно, ці вразливості в мережі блокчейн можуть бути використані для здійснення атак, таких як атаки подвійних витрат. У цьому типі атаки зловмисник надсилає транзакцію жертві, де щільність вузлів нижча, одночасно створюючи другу транзакцію, яку він надсилає в місце з вищою щільністю вузлів, де наступний блок з більшою ймовірністю буде підтверджений. В результаті, коли друга транзакція приймається великим відсотком мережі, перша транзакція, відправлена жертві, ігнорується мережею і не отримує підтвердження.

Запропоноване рішення проблеми розповсюдження блоків базується на оптимізації перевірки блоків або транзакцій. Розглянемо етап перевірки, який поділяється на три підетапи:

1. Перевірка складності: Це перевірка накопиченої складності, яка представляє собою обчислювальну потужність, що використовується при обчисленні хешів блоків в цьому ланцюжку.
2. Перевірка транзакцій: Передбачає перевірку транзакцій, включених в блок.
3. Завершення: Це завершальний етап, на якому процес верифікації завершується.

Перевірка складності займає менше часу, ніж перевірка транзакцій. Тому пропонується надсилати ініціююче повідомлення для обміну після перевірки складності, дозволяючи більш трудомісткій перевірці транзакцій продовжуватися, не затримуючи решту процесу.

Існує також другий підхід до обміну даними. Після отримання запрошення на обмін, запрошення надсилається всім сусіднім вузлам разом із запитом на дані. Будується черга для визначення пріоритету відправки транзакції, і після перевірки складності блок або транзакція відправляється відповідно до встановленої черги.

Третій підхід пропонує змінити топологію мережі на зіркоподібну підмережу. Хоча цей метод збільшує швидкість транзакцій, оскільки кожна транзакція буде проходити тільки через один вузол, він суперечить основній концепції криптовалют і децентралізованих мереж в цілому - децентралізації. Крім того, це значно знижує відмовостійкість мережі, оскільки якщо центральний вузол виходить з ладу, вся мережа може бути скомпрометована.

**РОЗДІЛ 5**

# Доказ парадоксу Пуассона для PoW

### **Гіпотеза**

Більшість користувачів очікують підтвердження блоку більше ніж 10 хвилин, не дивлячись на те, що в середньому воно становить 10 хвилин.

### **Технічне завдання на розроблення моделі**

Блок, моделюється як процес Пуассона зі швидкістю λ. Тут середній час між блоками, , встановлюється на основі цільової складності в операції майнінгу PoW; для Bitcoin = 10 хвилин. Пуассонівський процес — це процес, у якому нові події (або надходження) відбуваються через випадкові проміжки часу після експоненціального розподілу. Крім того, інтервали між будь-якими двома подіями не залежать один від одного та статистично ідентичні. Нагадаємо, що експоненціальна випадкова величина X з параметром λ має розподіл

Також нагадаємо, що пуассонівська випадкова величина Y з параметром λ має розподіл

У пуассонівському процесі кількість подій на інтервалі довжиною T є пуассонівською випадковою величиною з параметром λT. Крім того, кількість подій у непересічних проміжках часу не залежить. Якщо ми розглядаємо малі інтервали (), то в інтервалі з ймовірністю λT є одна

подія і жодної іншої. Таким чином, процес Пуассона можна імітувати шляхом «гри в орлянку» де послідовність (незалежних) підкидань монети має дуже малу вірогідність випаду «орла». Тепер ми розуміємо, чому процес майнингу має таку властивість.

Уявіть, що кількість майнерів у системі та загальна обчислювальна потужність у їхньому розпорядженні є постійною протягом певного періоду часу. Припустімо, що кожен з комп’ютерів майнерів безперервно перебирає хеші, і це єдине обчислення, яке вони виконують. Тоді можна

сказати, що загальна кількість хешів, які обчислюються (еквівалентно, загальна кількість оказій, які перебираються) за одиницю часу є приблизно постійною. Скажімо, щосекунди обчислюється мільярд хешів.

Крім того, припустимо, що рівень складності хеш-головоломки для кандидатного блоку дуже висока. Наприклад, скажимо, що перші тридцять п’ять бітів хешу мають бути нульовими, щоб блок був дійсним. Імовірність того, що певний nonce відповідатиме цьому критерію, становить 2−35 ≈ 3 × 10−11. Таким чином, ймовірність того, що хеш-головоломка буде розв’язана будь-яким майнером за певну секунду часу, становить 0,03 , невелике число (ми припустили, що кожну секунду обчислюється мільярд хешів). Те, чи знайдено хеш підтвердження роботи в певну секунду, не впливає на те, чи буде він знайдений у наступну секунду. Причина цього в тому, що хеш-функція, по суті, є випадковим оракулом; хеш-значення для різних вхідних даних не залежать одне від одного. Таким чином, процес видобутку добре моделюється як процес Пуассона. При моделюванні процесу видобутку як пуассонівського процесу ми зосереджуємося лише на моменті створення нових дійсних блоків.

Модель процесу Пуассона актуальна незалежно від кількості майнерів, їхньої індивідуальної обчислювальної потужності, того, чи працюють різні майнери над одним або різними блоками, і коли різні користувачі отримують щойно видобуті блоки. Параметр λ процесу майнінгу, який називається швидкістю майнінгу, дорівнює середній кількості блоків, видобутих за одиницю часу. У біткоїнах λ дорівнює 1/(600 с), тобто один блок кожні 600 секунд (десять хвилин).

З варіаціями загальної обчислювальної потужності припущення про фіксовану швидкість майнінгу спростовуються. Насправді загальна обчислювальна потужність не змінюється раптово. Таким чином, протягом невеликого періоду часу швидкість є приблизно постійною. Регулювання параметра складності через регулярні проміжки часу допомагає підтримувати швидкість видобутку на одному рівні.

### **Математичний опис моделі**

Чи є надходження блоків Пуассонівським процесом. Спробуємо довести це. Відразу відмітимо що незалежно від того, чи моделюється глобальний хешрейт H(t). емпірично або параметрично, для кожного входження блоку моделі в цьому моделюванні хешрейт визначається до вибірка випадкового процесу Xi(t).

* Для моделей із детермінованим налаштуванням складності, складність коригується в детермінований час моменту yn, які не відповідають випадковим моменти надходжень блоків та швидкість надходження блоків λ(t) не залежить від надходжень блоку в попередньому сегменти. Якщо немає затримки, то на кожному інтервалі модель є неоднорідним процесом Пуассона.
* Для моделей із випадковим налаштуванням складності, складність регулюється у випадкові моменти часу, після кожного 2016 сегменту блокчену використовуючи рівняння. Якщо немає затримки розповсюдження, тоді кожен сегмент процесу є неоднорідним пуассонівським зі швидкістю, заданою як λ(t) = H(t)/Di. Оскільки швидкість надходження блоку λ(t) залежить від першого та останнього надходження в попередній сегмент блокчейну, процес не Пуассонівський протягом послідовного періоду часу сегменту.
* Якщо присутня затримка поширення, то прибуття блоку процес навіть не є неоднорідним процесом Пуассона на одному сегменті. У наступному розділі ми

Порівняємо їх моделювання до даних про позначку часу з блокчейну біткоїна.

Отже, ми не будемо враховувати динамічну складність і змоделюємо сегмент з 2016 блоків, що дорівнює приблизно місяцю в реальному часі, або 20160 хвилин. Ми також припускаємо, що розподіл блоків у мережі відбувається миттєво, без затримок.

Точковий процес N є процесом Пуассона на , якщо він має наступні дві властивості.

1. Випадкова кількість точок N([a, b)) точкового процесу N, розташованих в обмеженому інтервалі [a, b) ⊂ R, є пуассонівською випадковою величиною із середнім Λ([a, b)), де Λ є невід’ємною мірою Радона.
2. Кількість точок точкового процесу N, розташованих на k інтервалах [a1, b1), . . . , [ak, bk) утворюють k незалежних пуассонівських випадкових величин із середніми Λ([a1, b1)), . . . ,Λ([ak, bk)).

Відтепер будемо записувати N([a, b)) як N(a, b) і Λ([a, b)) = Λ [a, b) для зручності. Перша властивість передбачає що

і , а друга властивість – це Основна причина придатності процесу точки Пуассона і зазвичай це основа статистичних тестів, які вимірюють адекватність моделей Пуассона. Розподіл Пуассона N(a, b) означає, що його дисперсія Var[N(a, b)] = Λ(a, b), факт який також використовується як статистичний тест. Міра Λ відома як міра інтенсивності або середнє значення міри процесу точки Пуассона. Припустимо, що a існує така функція λ(t), що

Тоді λ(t) визначена як функція швидкості. Якщо λ(t) є сталою λ > 0, то процес називається однорідним точковим процесом Пуассона. Інакше процес називають неоднорідним або **неоднорідним точковим процесом Пуассона**. Якщо обмежити нашу увагу інтервалом невід’ємних чисел [0, ∞), міра інтенсивності задається формулою

Для пуассонівського процесу N з мірою інтенсивності Λ ймовірність існування n точок в інтервалі [a, b) дорівнює

Час надходження та час між надходженнями: розглянемо точковий процес {X(i)}i≥1, визначений на невід’ємних дійсних числах із майже напевно кінцевою кількістю точок у будь-якому обмеженому інтервалі. Тоді ми можемо інтерпретувати точки процесу як часи добування нових блоків та розмістити їх у порядку зростання, X1 ≤ X2 ≤ . . .. Тоді відстані між сусідніми точками дорівнюють Ti := Xi − Xi−1 для i = 2, 3, . . . і T1 = X1. Випадкові величини Ti відомі як час очікування або час між надходженнями. Для однорідного процесу Пуассона зі швидкістю λ відповідні часи між надходженнями є незалежними та однаково розподіленими експоненціальними випадковими величинами із середнім значенням 1/λ

Де властивість експоненціального розподілу без пам’яті було використано. Це не стосується неоднорідного точкового процесу Пуассона з інтенсивністю λ(t), де перший час між надходженнями T1 = X1 має розподіл

За першого часу очікування T1 = t1 умовний розподіл другого часу очікування T2 є

і так далі для k ≥ 2

Можна показати, що k-й час надходження Xk має розподіл

З щільністю

Умова на n точок пуассонівського процесу, що існує в деякому обмеженому інтервалі [0, t]. Ми називаємо ці точки умовним часом надходження блоку. Якщо процес Пуассона є однорідним, то умовні часи надходження рівномірно і незалежно розподілені, утворюючи n рівномірних випадкових величин на [0, t]. Ця різниця між часом очікування Ti та умовним часом надходження Ui відіграє роль у тесті Пуассона.

Для неоднорідного пуассонівського процесу кожна точка Ui незалежно розподілена на інтервалі [0, t] із розподілом

,

Якщо розподіл кожного Ui відомий і оборотний, то кожен Ui може бути перетворений в рівномірну випадкову величину на [0, 1], що призводить до n незалежних рівномірних випадкових величин. Іншими словами, Λ(t) перетворює процес Пуассона на однорідний процес Пуассона з густиною один на відрізку дійсних чисел. Отже, статистичні методи для неоднорідних процесів Пуассона часто передбачають перетворення даних перед виконанням аналізу.

### **Аналіз результатів моделі**

Рисунок 10 Графік щільності часу в залежності від кількості транзакцій

Підводячи підсумок, швидкість видобутку біткоїна підпорядковується пуассонівському розподілу, що означає, що більшість блоків видобувається протягом 10-хвилинного періоду. Однак, враховуючи випадкову природу біткоїна, завжди будуть певні блоки, на видобуток яких майнерам знадобиться більше або менше часу. Отже, середній час підтвердження транзакції повинен коливатися в межах 10 хвилин, як було продемонстровано вище.

Парадокс - це твердження, яке на перший погляд здається абсурдним, але при більш детальному розгляді виявляється одночасно обґрунтованим і суперечливим. Розглянемо, наприклад, припущення, що більшість людей очікують більшого часу підтвердження транзакції, хоча в середньому він становить 10 хвилин.

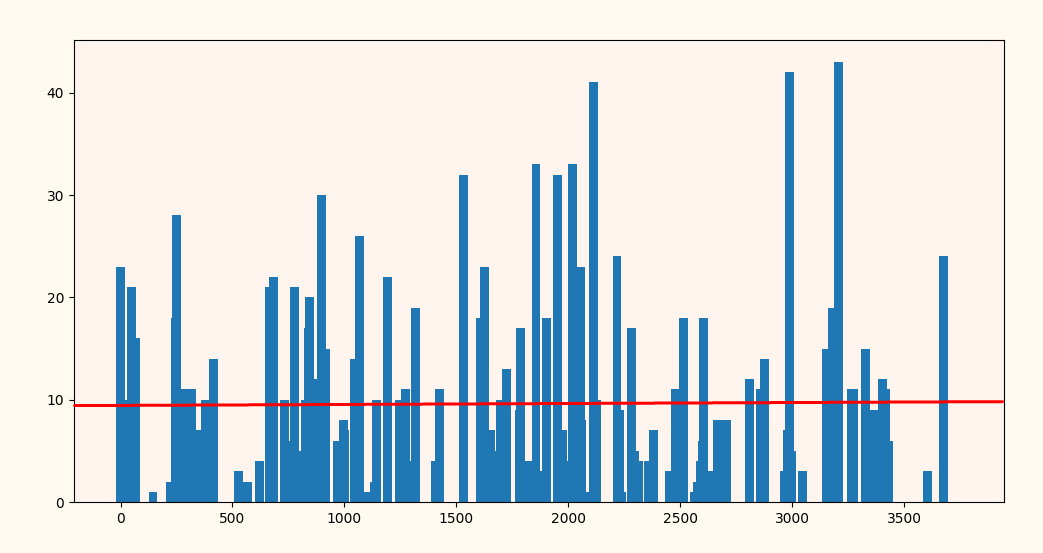
Щоб перевірити цю гіпотезу, ми використали невелику вибірку з 140 блоків біткоїнів з номерами від 759149 до 759289.

Рисунок 11 Діаграма кількості транзакцій в залежності від часу підтвердження блоку

Наступна гістограма ілюструє, що час підтвердження транзакцій дійсно підпорядковується пуассонівському розподілу. Більшість транзакцій, 78%, підтверджуються між 5 і 20 хвилинами. Середній час пошуку блоку становить близько 9,9 хвилин.

На графіку нижче показано хвилини між блоками, представлені помаранчевою лінією. Обсяг транзакцій за блок зображено синіми стовпчиками. Наші результати показують, що середня кількість транзакцій за блок становить приблизно 1805.



Рисунок 12 Середня кількість транзакцій на бло

Синя лінія відображає час, витрачений на очікування підтвердження блоку. Вона показує зв'язок між часом очікування блоку і обсягом транзакцій в наступному блоці. По суті, якщо підтвердження відбувається швидко, як показано на 10.47, наступний блок буде порожнім. Навіть якщо 2000 транзакцій було підтверджено протягом 5 хвилин, блоки, що їх містили, були заповнені лише наполовину, що свідчить про те, що менше користувачів отримали швидке підтвердження порівняно з користувачами о 13.21, коли майже 11 000 транзакцій чекали на підтвердження 40 хвилин.

Таким чином, не обов'язково, що всі блоки, які отримали швидке підтвердження, є невеликими. Наприклад, о 10:55 один блок потребував 5 хвилин майнінгу, але підтвердив лише 200 транзакцій. І навпаки, о 15:28 блок був виявлений за 0 хвилин і містив 2982 транзакції. Отже, розмір блоку залежить від кількості транзакцій, що очікують підтвердження в пулі пам'яті (перевантаження), а не від швидкості надходження блоків.

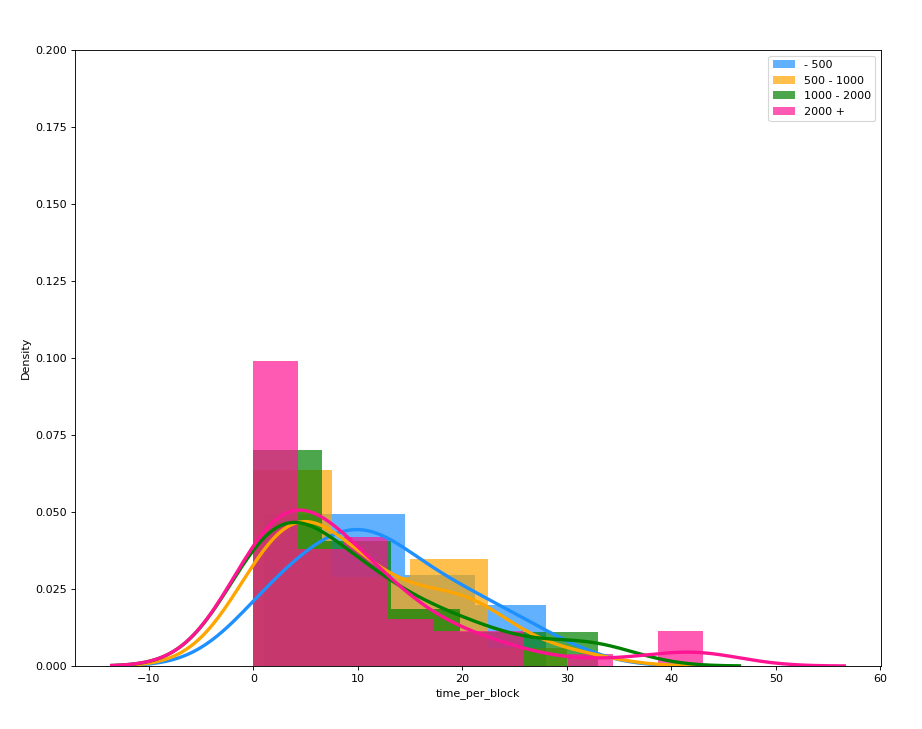
Хоча цей аналіз даних не є остаточним доказом існування парадоксу Пуассона в біткоїні, доказ насправді криється в початковій гістограмі, а саме в її довгому хвості. Більшість респондентів у нашій вибірці чекають на підтвердження більше 10 хвилин, хоча середній час очікування становить 9,9 хвилин. Це пов'язано з довгим правим хвостом розподілу Пуассона. Іншими словами, існує більше можливостей виявити блок між 10-40 хвилинами, оскільки цей часовий інтервал в чотири рази більший, ніж інтервал 0-10 хвилин. Для кращого розуміння зверніться до діаграми нижче.

Рисунок 13 Графік щільності часу в залежності від кількості транзакцій

|  |  |
| --- | --- |
| Час підтвердження блоку | Відсоток вибірки |
| 0 – 10 хвилин | 40 % |
| 10 – 40 хвилин | 60 % |

Таким чином, 2/5 нашої вибірки отримали підтвердження транзакції менш ніж за 10 хвилин, тоді як решта 3/5 були свідками того, що час підтвердження перевищував 10 хвилин. Це те, що ми називаємо парадоксом Пуассона.

Існує кілька інших факторів, які можуть призвести до того, що транзакції Bitcoin будуть довшими або коротшими за середній 10-хвилинний часовий проміжок. Однак вони не пов'язані безпосередньо з парадоксом Пуассона.

По-перше, майнери витягують і хешують транзакції з пулу пам'яті в заголовок блоку перед тим, як шукати наступне значення nonce. Це може призвести до відставання. Наприклад, нові транзакції, які майнери ще не забрали, залишаються в пулі пам'яті, поки не буде видобуто попередній блок. Це включає в себе час, необхідний для виявлення нових блоків. Транзакції також можуть застрягти, якщо пов'язані з ними комісії занадто низькі.

Тим не менш, якщо транзакція застрягла, її комісію можна збільшити за допомогою таких методів, як Child-Pays-for-Parent (CPFP) або Replace-by-Fee (RBF). Збільшення комісії за транзакцію підвищує її пріоритет, що підвищує ймовірність того, що майнери включать її в наступний блок. Крім того, деякі майнінг-пули мають можливість додавати транзакції безпосередньо до своїх блоків, що може прискорити час проведення транзакцій.

### **Висновок**

У цьому розділі нашого дослідження ми пропонуємо набір моделей, спрямованих на розуміння точкового процесу епох майнінгу блоків у блокчейні біткоїна. Дослідження виконано з використанням симуляцій та доступних даних з блокчейну. Однак це пов'язано з певними проблемами:

1. Глобально невідома швидкість хешування, яка диктує швидкість виявлення блоків;
2. Історично відоме, але випадкове значення складності майнінгу.

Крім того, незважаючи на те, що дані про час прибуття блоків є вичерпними, вони не є повністю достовірними. Ми пропонуємо модель точкового процесу, в якій процес надходження блоків імітує нерівномірний пуассонівський процес в періоди між змінами складності. Швидкість пропорційна відношенню швидкості хешування до складності, але зберігає незалежність при зміні складності. При глобальній швидкості хешування точка зміни складності (і, таким чином, зміна швидкості надходження блоків) визначається шляхом дискретизації процесу.

Тим не менш, глобальна швидкість хешування залишається високою, а механізм зворотного зв'язку щодо складності демонструє достатню затримку, що призводить до швидкості надходження блоків, яка приблизно на 11,5% перевищує базову швидкість в шість блоків на годину. Отже, і пропускна здатність транзакцій, і загальний дохід майнерів від бонусів перевищують базові прогнози.

Крім того, моменти, коли винагорода за майнінг блоків зменшується вдвічі і коли всі біткоїни будуть видобуті, настануть раніше, ніж якби блоки видобувалися зі швидкістю шість блоків на годину. Крім того, що ми запропонували модель процесу надходження блоків, ми встановили взаємозв'язок між частотою надходження блоків і експоненціальним зростанням швидкості хешування. Ми також запропонували практичне наближення, яке ілюструє поведінку границі, незалежно від початкових умов і збурень процесу надходження блоків. Це наближення було підтверджено за допомогою симуляцій та вимірювань з блокчейну.

Ми також підтвердили гіпотезу про парадокс Пуассона під час очікування підтвердження транзакції. Оскільки розподіл має довгий правий хвіст, більшість користувачів чекають довше, ніж середній час підтвердження блоку, в той час як меншість користувачів користуються швидшою швидкістю.

**РОЗДІЛ 6**

# Реалізація програмного забезпечення

У цьому розділі було обговорено дизайн, архітектуру та реалізацію нашого програмного забезпечення на основі блокчейну.

### **Створення програмного продукту**

Програмний продукт було створено з використанням мови програмування Go, завдяки її простоті, високій продуктивності та потужній підтримці паралельних процесів, що є критично важливим для роботи мережі блокчейн.

Ми також використали різні пакети Go, такі як bufio, fmt, os, strconv, strings, json, blockchain і network, щоб абстрагуватися і спростити деякі завдання, пов'язані з мережею, криптографією, взаємодією користувачів і управлінням даними.

### **Класи та об'єкти**

Програмне забезпечення використовує декілька класів та об'єктів, визначених у пакетах блокчейну та мережі. Короткий опис кожного з них наведено нижче:

* ‘User’: Клас User представляє користувача в мережі блокчейн. Він надає методи для доступу до специфічної для користувача інформації, такої як адреса та гаманець користувача. Об'єкти цього класу створюються методами userNew або userLoad, в залежності від наданих аргументів.
* ‘Transaction’: Клас Transaction представляє транзакцію в блокчейні. Він інкапсулює всі необхідні деталі транзакції, такі як відправник, одержувач і сума переказу.
* ‘Package’: Клас Package з мережевого пакету представляє пакет даних, який надсилається через мережу. Він включає опцію (операцію, яку потрібно виконати) і дані (пов'язані з нею дані).

### **Реалізація інтерфейсу користувача**

Інтерфейс користувача для програмного забезпечення - це інтерфейс командного рядка (CLI). Таке рішення було прийнято для того, щоб зробити додаток легким і незалежним від платформи.

CLI надає користувачеві команди для взаємодії з мережею блокчейн, такі як створення нового користувача, завантаження існуючого користувача, створення транзакцій і запит балансів. Кожна команда має структуру /command arg1 arg2 ..., де command - ім'я команди, а arg1, arg2, ... - аргументи команди.

Ось приклад коду функції розбору та обробки команд:

**func** handleClientInput() {

// ...

**for** {

message := inputString("> ")

splitted := strings.Split(message, " ")

**switch** splitted[**0**] {

**case** "/exit":

os.Exit(**0**)

**case** "/user":

handleUserCommand(splitted)

**case** "/chain":

handleChainCommand(splitted)

**default**:

fmt.Println("Undefined command")

}

}

}

### **Реалізація програмного модуля**

Основна логіка програми інкапсульована в головній функції handleClientInput. Ця функція обробляє команди користувача, викликає відповідні функції на основі команд і обробляє будь-які помилки, які можуть виникнути під час цього процесу.

Програма використовує функцію init для розбору аргументів командного рядка та ініціалізації глобальних змінних Address і User. Потім функція main викликає функцію handleClientInput для обробки команд клієнта.

### **Інструкція для користувача програми**

Щоб скористатися програмою, виконайте наступні кроки:

* Запустіть виконуваний файл з відповідними аргументами командного рядка: -loadaddr:<шлях до файлу адреси>, -newuser:<ім'я користувача> або -loaduser:<ім'я користувача>.
* Після запуску програми ви побачите запрошення >. Тут ви можете ввести наступні команди:
  + /адреса користувача: Вивести адресу поточного користувача.
  + /гаманець користувача: Надрукувати гаманець поточного користувача.
  + /user balance: Роздрукувати баланс поточного користувача.
  + /chain print: Вивести поточний стан блокчейну.
  + /chain tx <отримувач> <сума>: Створити транзакцію для відправки монет на вказану суму одержувачу.
  + /chain balance <user>: Вивести баланс користувача.
  + /exit: Вийти з програми.
* Після завершення роботи використовуйте команду /exit для безпечного виходу з програми.

### **Детальний опис класів та об'єктів**

Давайте розширимо наше дослідження класів та об'єктів, реалізованих у цьому програмному забезпеченні.

**Класи Block і BlockChain**

Клас Block представляє окремий блок у блокчейні. Кожен блок включає в себе CurrHash, карту Mapping (яка відображає стан кожної адреси в блоці), Miner (хто видобув цей блок) і TimeStamp (коли цей блок був створений).

Клас BlockChain представляє весь блокчейн і включає в себе БД для підключення до бази даних та індекс для відстеження довжини блокчейну. Він надає різні методи для маніпуляцій та запитів до блокчейну, такі як LastHash, Balance, Size, AddBlock та HeadBlock. Ці методи дозволяють нам взаємодіяти з блокчейном та його окремими блоками.

Ось приклад реалізації класу BlockChain:

**type** BlockChain **struct** {

DB \*sql.DB

index **uint64**

}

**func** (chain \*BlockChain) AddBlock(block \*Block) **error** {

chain.index++

\_, err := chain.DB.Exec("INSERT INTO BlockChain (Hash, Block) VALUES (?, ?)",

Base64Encode(block.CurrHash),

SerializeBlock(block))

**return** err

}

**Клас Transaction**

Клас Transaction, який представляє транзакцію в блокчейні, є критично важливим для функціонування нашого блокчейн-додатку. Він включає різні поля, такі як заголовок (який містить мета-інформацію про транзакцію) та входи і виходи (які містять фактичні дані транзакції).

Ось спрощений огляд класу Transaction:

**type** Transaction **struct** {

Header TransactionHeader

Inputs []TxInput

Outputs []TxOutput

}

**Реалізація мережевого пакету**

Мережевий компонент нашого програмного забезпечення реалізований за допомогою мережевого пакету. Цей пакет реалізує мережевий зв'язок між вузлами в мережі блокчейн.

Мережевий пакет включає клас Package, який інкапсулює дані, що надсилаються мережею, та функцію Send, яка надсилає пакет на вказану адресу.

Ось спрощений огляд класу Package та функції Send:

**type** Package **struct** {

Option **byte**

Data **string**

}

**func** Send(address **string**, pkg \*Package) \*Package {

// implementation omitted for brevity

}

Команди запитів та маніпуляцій з блокчейном

Наше програмне забезпечення надає різні команди для взаємодії з блокчейном, такі як /chain tx для створення транзакції та /chain balance для запиту балансу користувача. Ці команди взаємодіють з блокчейном шляхом виклику методів в об'єкті BlockChain.

Наприклад, ось як реалізована команда /chain balance:

**func** chainBalance(splited []**string**) {

**if** len(splited) != **2** {

fmt.Println("len(splited) != 2\n")

**return**

}

printBalance(splited[**1**])

}

**func** printBalance(useraddr **string**) {

**for** \_, addr := **range** Address {

res := nt.Send(addr, &nt.Package{

Option: GET\_BLNCE,

Data: useraddr,

})

**if** res == **nil** {

**continue**

}

fmt.Printf("Balance (%s): %s coins\n", addr, res.Data)

}

fmt.Println()

}

У цій реалізації printBalance зв'язується з кожним вузлом мережі блокчейн (представленим зрізом Address), щоб запитати баланс за вказаною адресою. Це робиться шляхом надсилання пакету GET\_BLNCE до кожного вузла і роздруківки відповідей.

1. Криптографічна хеш-функція — це одностороння функція, яка приймає рядок довільної довжини і повертає рядок фіксованої довжини. Властивість однобокості говорить про те, що з x [y = f(x)] легко отримати y, але важко отримати x з y [x = f-1(y)]. В якості криптографічних хеш-функцій рекомендується використовувати хеш-функції сімейства SHA-2 (sha224, sha256, sha384, sha512) або SHA-3 (keccak). [3, с.111] стверджує, що хешфукції «в сирому вигляді» мають недоліки, схожість подовження повідомлень і колізій. Позбутися від другого проблематично, через існування атаки на день народження, тим самим прийнято вибирати хеш-функції більшого порядку. Позбутися від першого недоліку, як приклад, можна, застосувавши до функції алгоритм HMAC. Але в блокчейні така атака буде марною, так як вимагає редагування вже існуючих даних і хешу даних, що тягне за собою необхідність редагування підпису, через подальші розбіжності між хешем і підписом. [↑](#footnote-ref-1)
2. Цифровий підпис є елементом асиметричної криптографії. В алгоритмах шифрування, таких як RSA, результатом є функція дешифрування 14, а перевірка підпису виконується за допомогою функції шифрування. Ідея полягає в тому, що функція дешифрування відома тільки творцеві закритого ключа, відповідно, тільки він здатний підтвердити свої дії за допомогою цієї властивості. [↑](#footnote-ref-2)
3. PoW (Proof-of-work) - доведення роботи, здійснюване шляхом вирішення складної математичної задачі (наприклад, знаходження шуканого хешу). Крім PoW, існують і інші методи доказування, такі як PoS (Proof-of-stake), PoA (Proof-of-authority), PoB (Proof-of-burn) і т.д. [↑](#footnote-ref-3)
4. Розрахувати поточну потужність майнерів можна за допомогою часових позначок. Якщо раніше видобуток займав у середньому 10 хвилин, а зараз 5, то можна стверджувати, що потужність видобутку зросла вдвічі. При цьому тенденція до зниження необхідного часу позначається несприятливим чином, так як почастішали випадки появи soft-fork. Через це блокчейни попередньо встановлюють функцію, яка регулює поточну складність блокчейна, підлаштовуючись під постійний час (наприклад, 10 хвилин). [↑](#footnote-ref-4)