Лабораторна робота №2: Механізми консенсусу та мережева архітектура

1. Мета роботи

Вивчити принципи досягнення узгодженості в децентралізованих системах, опанувати механізми Proof of Work (PoW) та Proof of Stake (PoS), а також зрозуміти особливості P2P-взаємодії та форків.

2. Теоретична частина

2.1. Проблема візантійських генералів

У розподілених системах головним викликом є досягнення консенсусу між вузлами, які не довіряють один одному. **Суть задачі:** Група генералів оточила вороже місто. Вони повинні домовитися про спільну дію (атака або відступ). Проблема в тому, що серед генералів можуть бути зрадники, які надсилатимуть суперечливі накази різним частинам армії, щоб зірвати операцію. **У контексті блокчейну:** Генерали — це вузли мережі, а зрадники — зловмисні вузли або технічні збої. Досягнення **Byzantine Fault Tolerance (BFT)** означає, що мережа продовжує працювати правильно, навіть якщо частина вузлів діє злонавмисно або офлайн.

2.2. Proof of Work (PoW) — Доказ роботи

PoW — це перший алгоритм консенсусу (використовується в Bitcoin).

- **Майнінг:** Процес пошуку спеціального числа (**nonce**), яке разом із даними блоку дає хеш, що задовольняє вимогу складності.
- **Складність (Difficulty):** Параметр, що визначає, скільки нулів має бути на початку хешу. Мережа автоматично адаптує складність (наприклад, кожні 2016 блоків у Bitcoin), щоб середній час створення блоку залишався стабільним.
- **Атака 51%:** Ситуація, коли один суб'єкт контролює понад 50% обчислювальної потужності мережі. Це дозволяє йому здійснювати подвійні витрати (double-spending) або цензурувати транзакції.

2.3. Proof of Stake (PoS) — Доказ частки володіння

Енергоефективна альтернатива PoW (Ethereum 2.0, Cardano, Solana).

- **Стейкінг:** Замість купівлі обладнання користувачі "заморожують" свої монети як заставу.
- **Валідатори:** Вузли, що обираються для створення блоку на основі кількості монет у стейку та часу їх утримання.

- **Slashing (Штрафування):** Механізм покарання, за якого валідатор втрачає частину стейку за спроби обману мережі або тривалий простій (downtime).
- **Перевага:** Не потребує гігантських витрат електроенергії та спеціалізованих чіпів (ASIC).

2.4. P2P-архітектура та механізм Gossip

У блокчейні немає центрального сервера. Вузли (peers) взаємодіють напряму.

- **Gossip Protocol (Протокол пліток):** Коли вузол отримує нову інформацію (транзакцію або блок), він передає її кільком випадковим сусідам. Ті, у свою чергу, передають далі. Це дозволяє інформації поширитися всією глобальною мережею за лічені секунди.

2.5. Форки та масштабованість

- **Soft Fork (М'який форк):** Оновлення, сумісне з попередніми версіями. Старі вузли приймають блоки нових вузлів.
- **Hard Fork (Жорсткий форк):** Кардинальна зміна протоколу, не сумісна зі старим ПЗ. Мережа розділяється на два окремі ланцюги (наприклад, Bitcoin та Bitcoin Cash).
- **Масштабованість:** Проблема обмеженої кількості транзакцій на секунду (TPS).
    - *Layer 1 (L1):* Збільшення розміру блоку або зміна алгоритму.
    - *Layer 2 (L2):* Рішення "понад" основним ланцюгом (наприклад, Lightning Network або Rollups).

3. Практична частина

Завдання 1: Реалізація алгоритму PoW

Вам необхідно доповнити функцію proof_of_work, щоб знайти nonce, який створює хеш із заданою кількістю нулів.

```
import hashlib

import time


def proof_of_work(block_data, difficulty):
    """
    block_data: рядок з даними блоку
```

```
    difficulty: кількість нулів на початку хешу
    """
    prefix = '0' * difficulty
    nonce = 0

    start_time = time.time()
    while True:
        # Створюємо рядок для хешування
        text = str(block_data) + str(nonce)
        hash_result = hashlib.sha256(text.encode()).hexdigest()

        # ПЕРЕВІРКА: чи починається хеш з потрібної кількості нулів?
        if hash_result.startswith(prefix):
            elapsed_time = time.time() - start_time
            return nonce, hash_result, elapsed_time

        nonce += 1

# Тестування
data = "Транзакція: А перевів Б 10 BTC"
for diff in range(1, 6):
    n, h, t = proof_of_work(data, diff)
    print(f"Складність {diff}: Nonce={n}, Час={t:.4f}с, Hash={h}")
```

Запустіть скрипт для складності від 1 до 6. Побудуйте графік залежності часу обчислення від складності.

Завдання 2: Моделювання вибору валідатора PoS

Створіть скрипт, який випадковим чином обирає валідатора, враховуючи їхню вагу (баланс).

```
import random
```

```
validators = {
    "Node_A": 100,  # 100 монет у стейку
    "Node_B": 500,  # 500 монет
    "Node_C": 50,   # 50 монет
    "Node_D": 350   # 350 монет
}

def select_validator(nodes_dict):
    # Реалізуйте логіку зваженого вибору
    elements = list(nodes_dict.keys())
    weights = list(nodes_dict.values())
    return random.choices(elements, weights=weights, k=1)[0]

# Проведіть 1000 симуляцій вибору та виведіть статистику
stats = {"Node_A": 0, "Node_B": 0, "Node_C": 0, "Node_D": 0}
for _ in range(1000):
    winner = select_validator(validators)
    stats[winner] += 1

print("Статистика виборів валідаторів:", stats)
```

4. Контрольні запитання

1. Чому в PoS не потрібне спеціалізоване обладнання (ASIC), на відміну від PoW?
2. Яким чином параметр difficulty у Bitcoin запобігає занадто швидкому випуску всіх монет?
3. Що таке "Finality" (остаточність) транзакції і як вона відрізняється в PoW та PoS?
4. Поясніть механіку "Slashing": за які саме дії валідатора можуть оштрафувати?

5. Що станеться з мережею, якщо понад 50% вузлів стануть зловмисними в системі з алгоритмом PoW?