

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Звіт до лабораторної №1
за темою:
Розгортання систем Ethereum та криптовалют

Оформлення звіту:
Юрчук Олексій, ФІ-52МН

19 лютого 2026 р.
м. Київ

ЗМІСТ

1 Невеличкий вступ	1
2 Огляд систем блокчейн і криптовалют	1
2.1 Bitcoin	1
2.2 Ethereum	1
2.3 Dash	2
2.4 NEO	2
2.5 Litecoin	3
3 Порівняльний аналіз архітектурних рішень та процедур розгортання	3
3.1 Механізми консенсусу	3
3.2 Архітектура мережі та P2P-взаємодія	3
3.3 Структури даних та управління станами	4
3.4 Можливості смарт-контрактів	5
3.5 Deployment Parameters and Requirements	5
3.6 Конфігурація генезис-блоку	5
3.7 Створення блоків і час їх генерації	6
3.8 Пропускна здатність	7
4 Взаємозамінність і сумісність модулів	7
4.1 Системні модулі і матриця сумісності блокчейнів	7
4.2 Висновки щодо взаємозамінності	8
4.2.1 Мережевий модуль (код)	8
4.2.2 Модуль узгодження (консенсусу)	8
4.2.3 Обробка транзакцій та модель станів	8
4.2.4 Модуль сховища	8
4.2.5 Криптографічні примітиви	8
4.3 Діаграма сумісності модулів	9
5 Розгортання приватної мережі Ethereum на основі WSL	9
5.1 Налаштування середовища і встановлення Geth	9
5.2 Конфігурація Genesis Block-у	11
5.3 Запуск приватної мережі	12
5.4 Спроба 2: Використання dev mode для швидкого розгортання	13
5.5 Спроба 3: Розгортання двох нод у приватній мережі для демонстрації P2P мережі	16
5.6 Проблеми та їх рішення	21
6 Висновки	22

1 Невеличкий вступ

Технологія блокчейн кардинально змінила наше уявлення про децентралізовані обчислення, фінансові транзакції та бездовірчі системи (trustless systems). З моменту публікації "Bitcoin whitepaper" [1] by Satoshi Nakamoto в 2008 році, з'явилося безліч платформ блокчейн, кожна з яких має свої архітектурні рішення, процедури розгортання та операційні характеристики.

У своїй лабораторній, я розглядав і порівнював, здебільшого зазначені в завданнях до лабораторної, особливості розгортання п'яти основних систем блокчейну/криптовалют: **Ethereum** [2, 3], **Bitcoin** [1], **Dash** [4], **NEO** [5] та **Litecoin** [6]. Свій аналіз я спрямував більше на практичні аспекти реалізації, розгортання та конфігурації кожної системи, вивчені їх базової архітектури, механізмів консенсусу, мережевих рівнів та можливості обміну модулями між ними.

Поки в планах наступне: у розділі 2 описати теоретичні основи кожної системи, далі в розділі 3 доволі детально порівняти їх за різними параметрами, розглянути їх взаємозамінність. А далі у розділі 4 планую навести покрокові, як я робитиму, практичне розгортання Ethereum (певно найпростіше, що можна взяти) у середовищі на базі WSL за setup-ами від [7, 8]. І відповідно в кінці підбити підсумки по отриманих результатах.

2 Огляд систем блокчейн і криптовалют

2.1 Bitcoin

Bitcoin – це певно перша і найвідоміша криптовалюта, опублікована в 2008 році анонімним девелопером з псевдонімом Сатоші Накамото. Вона використовує модель **UTXO (Unspent Transaction Output)** для відстеження балансів і механізм консенсусу **Proof-of-Work**, заснований на алгоритмі хешування SHA-256.

Ключові особливості Bitcoin включають в себе:

- **Жодних смарт-контрактів** (в традиційному сенсі) – Bitcoin Script навмисно є non-Turing-complete.
- **Block time** – генерація нового блоку ≈ 10 хвилин, **block size** обмежений $\approx 1\text{--}4$ MB (з SegWit).
- **Загальний обсяг** обмежений 21 мільйоном BTC.
- **Networking** – використосує особливий мережевий P2P protocol через TCP (port 8333), розповсюдження інформації на основі gossip-based ("пліткування з сусідами").

2.2 Ethereum

Ethereum – відкрита розподілена обчислювальна платформа на основі блокчейну, яка підтримує функціональність смарт-контрактів. На відміну від Bitcoin, який був розроблений в першу чергу як цифрова валюта, Ethereum був задуманий як універсальний програмований блокчейн (!). Власна криптовалюта блокчейну Ethereum це **Ether (ETH)**.

Основні архітектурні особливості Ethereum включають:

- **Ethereum Virtual Machine (EVM)** – віртуальна машина, яка виконує смарт-контракти, з повною функціональністю Тюрінга (це такий а-ля емулятор для виконання програм за скінчений час і пам'ять);
- **State model** – модель на основі облікових записів діє як банківська книга, відстежуючи баланс і забезпечує можливість укладання складних смарт-контрактів. (в Bitcoin застосовується UTXO model, де рахунки ефективно управляються смарт-контрактами, і в якій забезпечується вищий рівень конфіденційності та здійснюється паралельна обробка даних);
- **Protocol of Consensus** – первісно застосовувався Proof-of-Work (Ethash), але у вересні 22 зробили трансфер на Proof-of-Stake (Casper/Beacon chain);
- **Smart Contracts** – пишуться мовами Solidity, Vyper та деякими іншими мовами і є EVM-compatible;
- **Gas system** – обчислювальні витрати вимірюються за допомогою так званих тарифів "gas fees".

Ethereum – це peer-to-peer надбудова над базою інтернет протоколів TCP/IP [9]. Кожен вузол запускає копію блокчейну та бере участь у валідації та розповсюдженні блоків. Офіційний клієнт **Geth** (Go-Ethereum) реалізує протокол RLC (Remote Procedure Call) Node Discovery Protocol, що базується на DHT (Distributed Hash Table) типу Kademlia для виявлення однорангових вузлів (peers) [9].

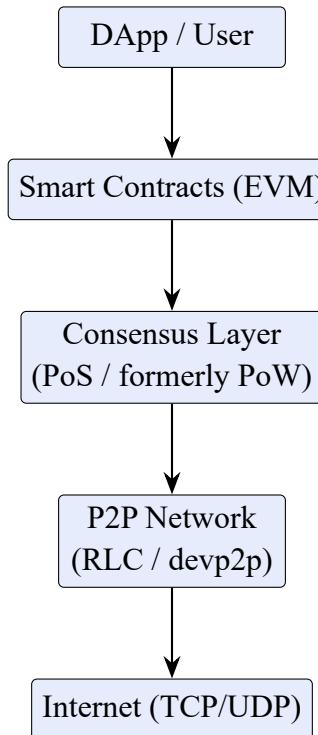


Рис. 1: Архітектура ”шарів” Ethereum-a

2.3 Dash

Dash (originally “Darkcoin”) являє собою fork від кодової бази Bitcoin, але з додатковими функціями конфіденційності та управління. В ньому було запроваджено:

- **Masternodes** – мережа другого рівня (second-tier network) з мотивованими (incentivized) вузлами, які забезпечують InstantSend і CoinJoin (PrivateSend).
- **Алгоритм гешування X11** – ланцюжкова послідовність із 11 криптографічних геш-функцій.
- **Система управління фінансами (Treasury system)** – 10% винагороди за блок виділяється на фінансування ідей щодо розвитку (за них голосуються masternodes).
- **Block time** – генерація нового блоку складає ≈ 2.5 хвилин.

2.4 NEO

NEO (колишня назва – AntShares) китайська блокчейн-платформа, яку частенько називають ”китайським ефіром (Ethereum)”. Вона також підтримує виконання смарт-контрактів, але має значні відмінності по своїй архітектурі:

- **Consensus** – делегована візантійська відмовостійкість (delegated Byzantine Fault Tolerance - dBFT), що забезпечує детермінованість виконання.
- **Dual token model** – NEO (токен управління, є неподільним) and GAS (токен-utilіта для виконання контрактів).
- **Multi-language support** – смарт-контракти можуть бути написані на різних мовах програмування, таких як: C#, Python, Java, Go (з подальшим виконанням у віртуальній машині NeoVM).
- **Інтеграція цифрової ідентичності** та орієнтація на суворе дотримання нормативних вимог.

2.5 Litecoin

Litecoin був створений Чарлі Лі в 2011 році як ”полегшена” версія Bitcoin. Це один з найперших форків від Bitcoin, який і досі має багато спільного з кодом Bitcoin. Основні особливості:

- **Scrypt hashing** – алгоритм, що вимагає великого обсягу пам’яті (первинно розроблявся для протидії майнінгу ASIC)
- **Block time** – генерування складає ≈ 2.5 хвилин (це у $4\times$ швидше за Bitcoin).
- **Total supply** – 84 мільйони LTC (у $4\times$ перевищує обсяг Bitcoin-a).
- **UTXO model** – така сама структура транзакцій, як у Bitcoin.
- **Test platform** – Часто стає тестовим майданчиком для функцій Bitcoin-a (наприклад, той же SegWit був активований спочатку на Litecoin).

3 Порівняльний аналіз архітектурних рішень та процедур розгортання

3.1 Механізми консенсусу

Механізм консенсусу є основним модулем, який визначає, як мережа блокчейну досягає згоди щодо стану реєстру. Кожен з названих нами п’яти блокчейнів використовують принципово різні підходи:

Блокчейн	Алгоритм гешування	Протокол	Остаточність
Ethereum	Gasper (PoS)	Proof-of-Stake	Probabilistic \rightarrow deterministic (2 epochs)
Bitcoin	SHA-256	Proof-of-Work	Probabilistic (~ 6 confirmations blocks)
Dash	X11 + Masternodes	Hybrid: PoW + PoS	Instant (via ChainLocks)
NEO	dBFT 2.0	Delegated BFT	Deterministic (1 block, no forks)
Litecoin	Scrypt	Proof-of-Work	Probabilistic (~ 6 confirmations blocks)

Таблиця 1: Порівняння протоколів консенсусу

3.2 Архітектура мережі та P2P-взаємодія

Всі п’ять систем працюють як однорангові децентралізовані (peer-to-peer) мережі, побудовані поверх мережі Інтернет. Однак їхні протоколи виявлення однорангових вузлів і комунікації значно відрізняються.

Ethereum використовує набір протоколів devp2p, який включає протокол виявлення вузлів RLPx на основі модифікованого DHT Kademlia. Кожен вузол ідентифікується за допомогою ідентифікатора enode (хеш SHA3 його відкритого ключа). Реалізація Kademlia використовує метрику відстані на основі XOR з 256 сегментами, кожен з яких містить до 16 записів. Виявлення однорангових вузлів (peers) використовує чотири типи повідомлень UDP: ping, pong, findnode та neighbors. Передача даних відбувається через зашифровані TCP-з’єднання з використанням транспортного протоколу RLPx із використанням шифрування ECIES (Elliptic Curve Integrated Encryption Scheme).

Bitcoin i Litecoin використовують простіший P2P-протокол на основі обміну інформацією (gossip-based). Masternodes виявляють однорангові вузли за допомогою DNS-посівів і поширення повідомлень

addr. Протокол працює через TCP (Bitcoin на порту 8333, Litecoin на порту 9333). Структурованого DHT немає; натомість вузли підтримують базу даних **addrman** (address manager) відомих однорангових вузлів.

Dash розширює протокол P2P Bitcoin окремим masternode layer. Masternodes утворюють накладну мережу другого рівня з детермінованим порядком, що дозволяє використовувати такі функції, як InstantSend quorums і CoinJoin mixing rounds.

NEO використовує структурований механізм виявлення однорангових вузлів із seed nodes. Консенсус dBFT вимагає меншого набору вузлів консенсусу (наразі 7 в основній мережі), які ”спілкуються” через спеціальний канал консенсусу.

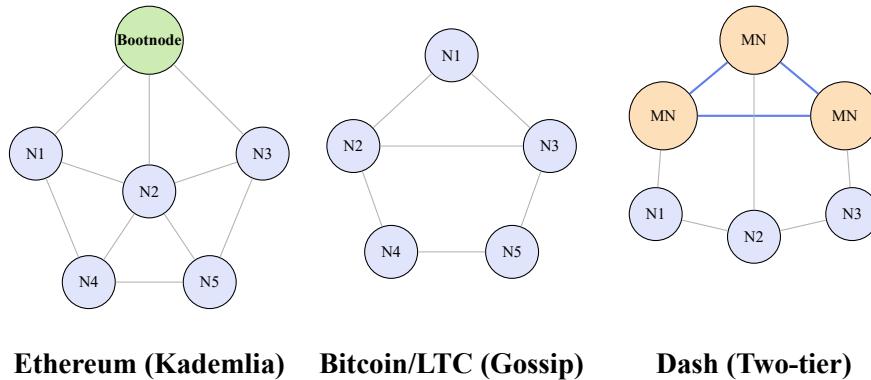


Рис. 2: Топології мереж P2P для різних блокчейнів

3.3 Структури даних та управління станами

Блокчейн	Станова модель	Структура даних	Сховище
Ethereum	Account-based	Modified Merkle-Patricia Trie	LevelDB / Pebble
Bitcoin	UTXO	Merkle Tree	LevelDB
Dash	UTXO (extended)	Merkle Tree + MN lists	LevelDB / RocksDB
NEO	Account-based	Merkle Tree + state root	LevelDB
Litecoin	UTXO	Merkle Tree	LevelDB

Таблиця 2: Станові моделі і структури даних в блокчейнах

Фундаментальна різниця між account-based системами (Ethereum, NEO) та UTXO (Bitcoin, Dash, Litecoin) проявляється в процесі розгортання та взаємозамінності модулів. У системах UTXO кожна транзакція споживає попередні виходи та створює нові. У account-based системах глобальний стан явно відстежує залишки на рахунках та сховище контрактів.

3.4 Можливості смарт-контрактів

Блокчейн	Віртуальне середовище	Мови програмування	Здібності блокчейну
Ethereum	EVM	Solidity, Vyper	Turing-complete
Bitcoin	Bitcoin Script	Script opcodes	Non-Turing-complete
Dash	—	—	No native smart contracts
NEO	NeoVM	C#, Python, Java, Go	Turing-complete
Litecoin	Bitcoin Script	Script opcodes	Non-Turing-complete

Таблиця 3: Порівняння можливостей блокчейнів щодо смарт-контрактів

3.5 Deployment Parameters and Requirements

В наступній таблиці хочу навести основні параметри розгортання для запуску повного вузла блокчейну:

Блокчейн	RAM (min)	ROM (min)	Default Port	Primary Client
Ethereum	8 GB	~1 TB+	30303	Geth / Nethermind
Bitcoin	2 GB	~600 GB	8333	Bitcoin Core
Dash	4 GB	~40 GB	9999	Dash Core
NEO	4 GB	~20 GB	10333	neo-cli (C#)
Litecoin	2 GB	~120 GB	9333	Litecoin Core

Таблиця 4: Параметри розгортання mainnet у 2024–2025 роках

3.6 Конфігурація генезис-блоку

Будь який блокчейн починається з так званого **генезис-блоку (genesis block)**. У приватних/тестових розгортаннях його конфігурація визначає початкові параметри мережі. Нижче наведено приклад для Ethereum (`genesis.json`):

```
{  
  "config": {  
    "chainId": 12345,  
    "homesteadBlock": 0,  
    "eip150Block": 0,  
    "eip155Block": 0,  
    "eip158Block": 0,  
    "byzantiumBlock": 0,  
    "constantinopleBlock": 0,  
    "petersburgBlock": 0,  
    "istanbulBlock": 0,  
    "berlinBlock": 0,  
    "londonBlock": 0  
  },
```

Основними важливими полями є:

- **chainId** – унікальний ідентифікатор, що відрізняє створювану приватну мережу від основної мережі (`chainId=1`).
 - **difficulty** – контролює складність майнінгової головоломки (mining puzzle complexity); нижчі значення дозволяють швидший майнінг у тестових середовищах.
 - **gasLimit** – максимальний дозволений обсяг гасу на блок, що встановлює верхню межу для виконання контракту.
 - **alloc** – дозволяє ініціалізувати (робити pre-funding) рахунків валютою Ether при їх створенні.
 - Поля ***Block** вказують, під яким номером блоку активуються різні EIP (пропозиції щодо вдосконалення Ethereum) та хард-форки. Встановлення всіх значень на 0 дозволяє активувати всі функції з самого початку.

Для порівняння, генезисний блок Bitcoin hard-coded в клієнтському програмному забезпеченні і не може бути налаштований таким же чином. Тому при розгортанні тестової мережі Bitcoin або мережі regtest параметри модифікуються під час компіляції за допомогою констант або пропорців конфігурації (`-regtest`, `-testnet`).

3.7 Створення блоків і час їх генерації

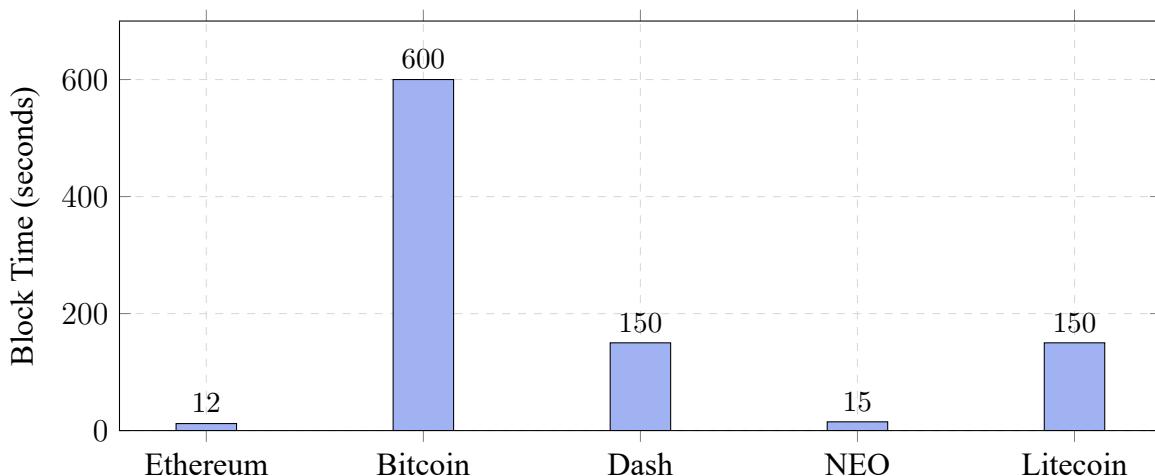


Рис. 3: Середній час генерації блоків (в секундах)

3.8 Пропускна здатність

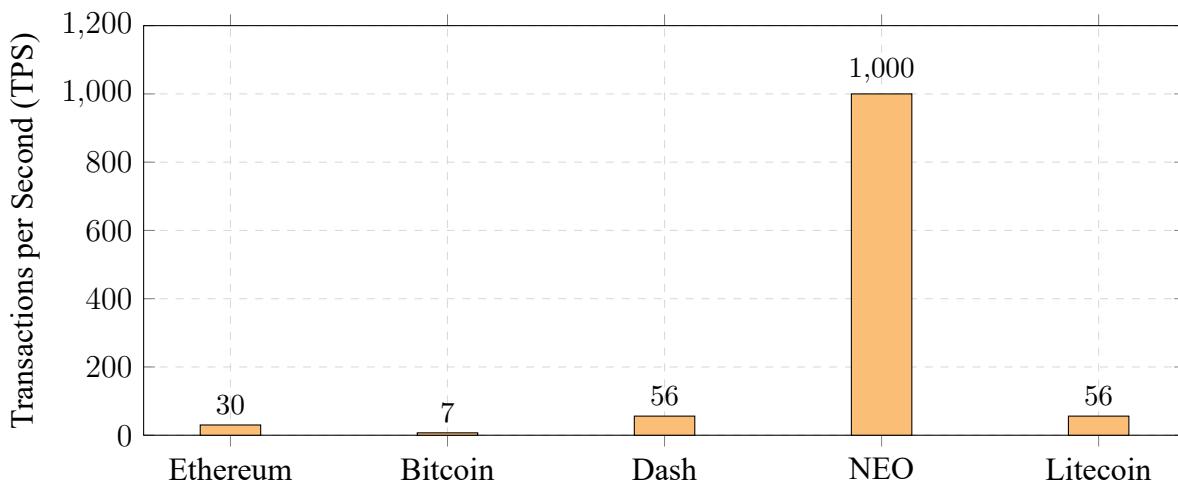


Рис. 4: Приблизна пропускна здатність транзакцій базового рівня (TPS) для різних блокчейнів

4 Взаємозамінність і сумісність модулів

Також я хочу розглянути таке питання: чи можна компоненти (модулі) однієї системи замінити на компоненти іншої системи, а також подивитися на архітектурну сумісність або несумісність.

4.1 Системні модулі і матриця сумісності блокчайнів

Блокчейн може бути розкладений на такі логічні модулі:

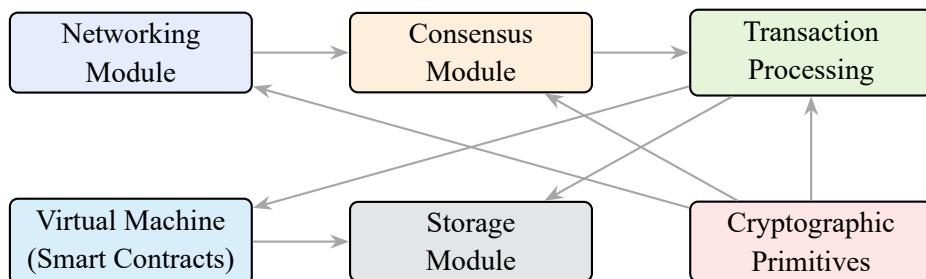


Рис. 5: Логічна декомпозиція модулів блокчейн-системи

Module	$\text{ETH} \leftrightarrow \text{BTC}$	$\text{BTC} \leftrightarrow \text{LTC}$	$\text{BTC} \leftrightarrow \text{Dash}$	$\text{ETH} \leftrightarrow \text{NEO}$	$\text{Dash} \leftrightarrow \text{LTC}$
Networking Code	Partial	High	High	Low	High
Consensus	No	No	No	No	No
Transaction Model	No	Yes*	Yes*	Partial	Yes*
Storage (LevelDB)	Yes	Yes	Yes	Yes	Yes
Cryptographic Primitives	Partial	Partial	Partial	Partial	Partial

Таблиця 5: Взаємозамінність модулів між системами

Примітка: "Так*" означає структурну сумісність із налаштуванням параметрів.

4.2 Висновки щодо взаємозамінності

4.2.1 Мережевий модуль (код)

Bitcoin, Litecoin і Dash мають дуже схожий протокол P2P gossip, оскільки Litecoin і Dash є прямими форками кодової бази Bitcoin. Їхні мережеві рівні є в основному взаємозамінними з незначними змінами параметрів (номери портів, рядки версій протоколу, «магічні байти»).

Ethereum використовує повністю інший стек протоколів (devp2p/RLPx з Kademlia DHT), що робить його мережевий модуль несумісним із системами сімейства Bitcoin. NEO також використовує власну реалізацію P2P, яка відрізняється від усіх перерахованих вище блокчайн систем, тому його мережевий код несумісний ні з Ethereum, так і з Bitcoin.

4.2.2 Модуль узгодження (консенсусу)

Модуль консенсусу є **фундаментально не взаємозамінним** у всіх системах. Механізм консенсусу кожної системи глибоко інтегрований з її структурою блоків, правилами переходу між станами, розподілом винагород і fork-choice algorithms. Наприклад:

- Gasper PoS (Ethereum) вимагає validator deposits, slashing conditions та epoch-based finality – концепцій, яких немає в чистому PoW від Bitcoin.
- dBFT у NEO вимагає фіксованого набору вузлів консенсусу з візантійською згодою, що архітектурно несумісно з будь-якою системою PoW.
- Masternode-based консенсус Dash додає рівень управління, якого немає в інших системах.

4.2.3 Обробка транзакцій та модель станів

Системи на основі UTXO (Bitcoin, Litecoin, Dash) мають структурно схожий формат транзакцій. Обробка транзакцій, наприклад, в Litecoin, в принципі, може бути використана в Bitcoin з невеликим коригуванням викликів хеш-функцій і параметрів серіалізації (serialization parameters). Однак заміна моделі Ethereum, заснованої на рахунках, на модель UTXO (або навпаки) вимагала б повного перепроектування управління станами, взаємодії з VM і самої логіки перевірки транзакцій.

4.2.4 Модуль сховища

Всі п'ять систем використовують LevelDB (або сумісні з ним key-value stores) як базову базу даних. Це чи не єдиний взаємозамінний компонент, оскільки механізм зберігання даних відносно відокремлений від логіки блокчейну. Наприклад, заміна LevelDB на RocksDB є пошириеною оптимізацією, яка не впливає на protocol-level behavior.

4.2.5 Криптографічні примітиви

Всі ці системи використовують криптографію на основі еліптичних кривих (secp256k1) для цифрових підписів. Відрізняються використанням різних геш-функцій: Ethereum використовує Keccak-256 (SHA-3), Bitcoin і Litecoin використовують SHA-256 і RIPEMD-160 для адрес, Dash використовує X11, а NEO використовує взагалі власну схему адрес. Заміна однієї геш-функції на іншу технічно можлива, але це порушить всю логіку виведення адрес, перевірки і самого майнінгу блоків.

4.3 Діаграма сумісності модулів

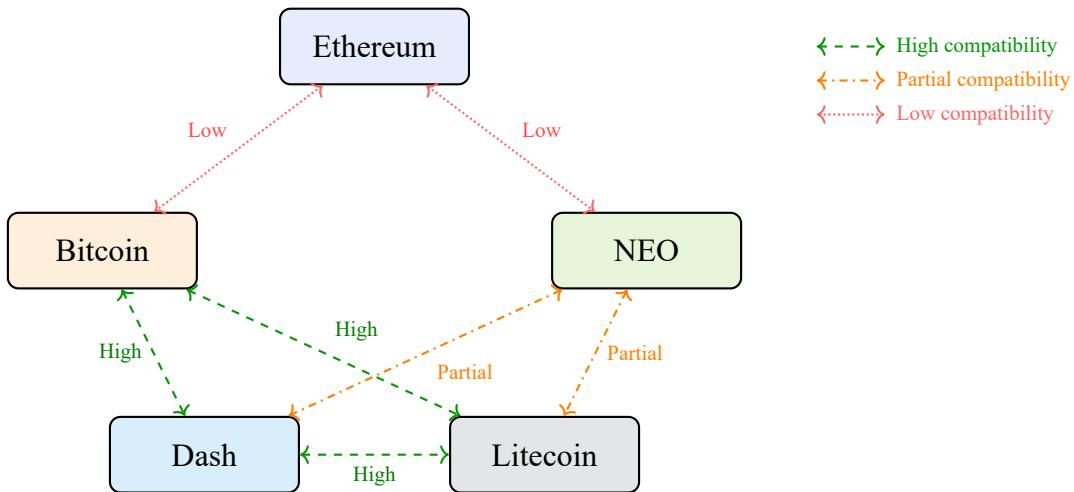


Рис. 6: Сумісність модулів між системами в цілому

Основні висновки:

- Системи, похідні від однієї кодової бази ($\text{Bitcoin} \rightarrow \text{Litecoin}$, $\text{Bitcoin} \rightarrow \text{Dash}$), мають найвищий ступінь взаємозамінності модулів, особливо в мережому модулі, обробці транзакцій та зберіганні даних.
- Ethereum та NEO, незважаючи на те, що обидві є платформами смарт-контрактів на основі облікових записів, вони використовують абсолютно різні мережеві стеки, реалізації VM та механізми консенсусу, що робить прямий обмін модулями непрактичним.
- Рівень зберігання даних** (LevelDB/RocksDB) є найбільш взаємозамінним компонентом, оскільки він працює нижче рівня самого протоколу обміну даними.
- Модулі консенсусу не є безпосередньо взаємозамінними(!).** Між будь-якою парою систем через тісний зв'язок із структурою блоків, переходами станів та дизайном економічних стимулів.

5 Розгортання приватної мережі Ethereum на основі WSL

Потужності моого комп’ютера не є великими (16 GB RAM, Windows 10, WSL v2.6.3), тому щоб спробувати відтворити блокчейн у себе локально, я вирішив для практичної частини взяти мережу Ethereum. Та й загалом причин вибору саме цієї системи декілька:

- Geth, що працює на базі Linux, повністю сумісний з WSL 2.
- Приватна мережа Ethereum вимагає мінімального дискового простору і може працювати з дуже низьким рівнем складності.
- Ethereum надає найширший набір функцій для тестування: рахунки, власне майнінг, перекази та розгортання смарт-контрактів.
- З 16 ГБ оперативної пам’яті система без проблем підтримує одночасну роботу декількох вузлів Geth.
- Багато докладних статей [7, 8], тому розібраться саме з Ethereum буде набагато простіше, чим читати документацію інших систем))

5.1 Налаштування середовища і встановлення Geth

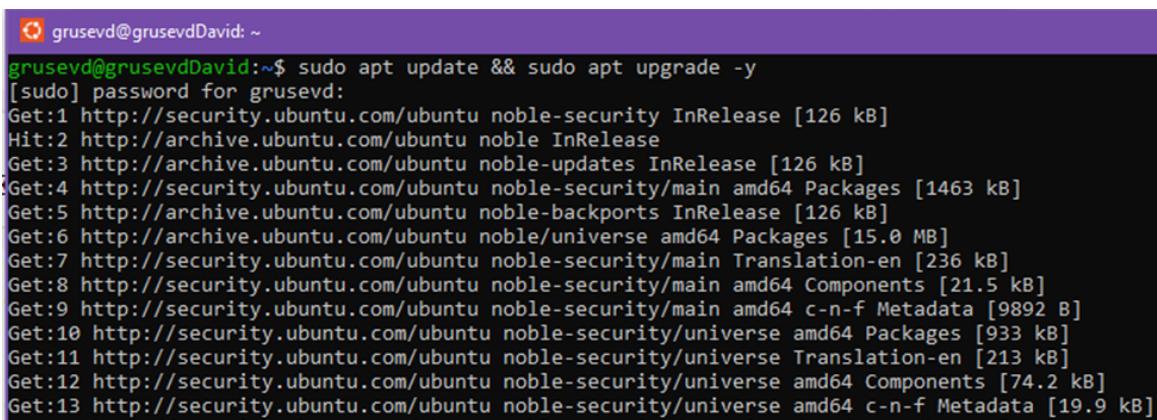
Крок 1: Оновити (а в кого нема – встановити) WSL та заінсталити усі необхідні залежності для роботи Geth.

Відкриваємо термінал WSL і запускаємо:

```
# Install Ubuntu distribution for WSL
wsl --install -d Ubuntu

# Update package list and upgrade existing packages
sudo apt update && sudo apt upgrade -y

# Install required tools
sudo apt install -y software-properties-common build-essential curl
→ wget
```



```
grusevd@grusevdDavid:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for grusevd:
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
:Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [1463 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [236 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [9892 B]
Get:10 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [933 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [213 kB]
Get:12 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [74.2 kB]
Get:13 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [19.9 kB]
```

Рис. 7: Ubuntu version output confirming successful installation

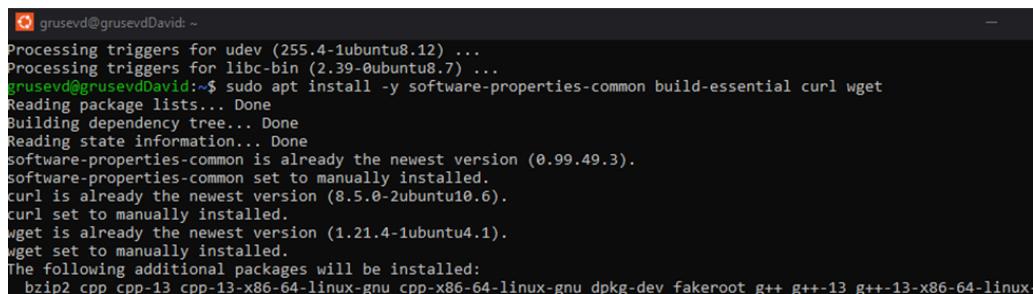
Крок 2: Встановлюємо Geth (Go-Ethereum)

```
# Add Ethereum PPA repository
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt update

# Install Geth and related tools
sudo apt install -y ethereum

# Verify installation
geth version
```

P.S. Можна було і по-старічковому просто скачати бінарник з сайту:
<https://geth.ethereum.org/downloads/>



```
Processing triggers for udev (255.4-1ubuntu8.12) ...
Processing triggers for libc-bin (2.39-0ubuntu8.7) ...
grusevd@grusevdDavid:~$ sudo apt install -y software-properties-common build-essential curl wget
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
software-properties-common is already the newest version (0.99.49.3).
software-properties-common set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
wget is already the newest version (1.21.4-1ubuntu4.1).
wget set to manually installed.
The following additional packages will be installed:
  bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13 g++-13-x86-64-linux-
```

Рис. 8: geth version output

Крок 3: Створюємо структуру робочої директорії для приватної мережі. Ми будемо запускати два вузли, тому створюємо папки для кожного з них, а також папку для логів.

```
# Create project directory  
mkdir -p ~/eth-private/{node1,node2,logs}  
cd ~/eth-private
```

5.2 Конфігурація Genesis Block-у

Крок 4: Створюємо genesis-файл [як потім вияволяється спроба №1] для нашої приватної мережі.

Рис. 9: genesis.json & it's configuration parameters.

Крок 5: Ініціалізуємо блокчейн для ноди 1.

```
geth --datadir ~/eth-private/node1 init ~/eth-private/genesis.json
```

Все як і очікували: "Successfully wrote genesis state".

```
grusev@grusevDavid:~/eth-private$ geth --datadir node1 init genesis.json
INFO [02-18 14:18:31.508] Maximum peer count                                ETH=50 total=50
INFO [02-18 14:18:31.510] Smartcard wallet not found, disabling directory
INFO [02-18 14:18:31.514] Set global gas cap                                     cap=50,000,000
INFO [02-18 14:18:31.515] Initializing the KZG library                           backend=gokzg
INFO [02-18 14:18:31.519] Defaulting to pebble as the backing database
INFO [02-18 14:18:31.519] Allocated cache and file handles
ta cache=512.00MiB handles=524,288
INFO [02-18 14:18:31.675] Opened ancient database
ta/ancient/chain readonly=false
INFO [02-18 14:18:31.675] Opened Era store
a/ancient/chain/era
INFO [02-18 14:18:31.675] State schema set to default
INFO [02-18 14:18:31.675] Load database journal from disk
WARN [02-18 14:18:31.675] Head block is not reachable
INFO [02-18 14:18:32.043] Opened ancient database
ta/ancient/state readonly=false
INFO [02-18 14:18:32.044] State snapshot generator is not found
INFO [02-18 14:18:32.044] Starting snapshot generation
dangling=0 elapsed="42.295us"
INFO [02-18 14:18:32.044] Initialized path database
B state-history="last 90000 blocks" journal-dir=/home/grusev/eth-private/node1/geth/chaindata
INFO [02-18 14:18:32.044] Writing custom genesis block
INFO [02-18 14:18:32.044] Resuming snapshot generation
dangling=0 elapsed="218.872us"
INFO [02-18 14:18:32.044] Generated snapshot
444.308us
INFO [02-18 14:18:32.044] Successfully wrote genesis state
grusev@grusevDavid:~/eth-private$
```

Рис. 10: `geth init` output confirming successful genesis state creation

5.3 Запуск приватної мережі

Крок 6: Запускаємо ноду 1

```
geth --datadir ~/eth-private/node1 \
--networkid 12345 \
--http \
--http.port 8545 \
```

```
--http.api eth,net,web3,miner,admin \
--nodiscover \
--port 30303 \
console 2>> ~/eth-private/logs/node1.log
```

Щось наплутав з конфігурацією власного Geth ланцюга і Geth повідомляє що ”цей ланцюг вже пройшов злиття”. Тож перестворимо з чистого каталогу без попередньо існуючих даних ланцюжка і у dev mode (мало бути на фото 11).

5.4 Спроба 2: Використання dev mode для швидкого розгортання

Прапор --dev Geth створює self-contained post-Merge development chain з pre-funded account і автоматичним блокуванням блоків. Це непоганий спосіб тестування рахунків, транзакцій та смарт-контрактів на сучасних версіях Geth, оскільки [як в процесі з’ясувалося] простір імен API `personal` та команди майнінгу PoW (`miner.start()`, `miner.setEtherbase()`) були видалені з Geth 1.12+.

Крок 7: Запускаємо ноду 1 у dev mode і подивимося її дані:

```
geth --datadir ~/eth-private/devnode \
--dev \
--http \
--http.port 8545 \
--http.api eth,net,web3,admin,debug \
--http.corsdomain "*" \
console 2>> ~/eth-private/logs/node1.log
```

Це відкриє нам консоль Geth JavaScript. Прапорець --dev створює приватний ланцюжок з pre-funded рахунком розробника і автоматично видобуває блоки при поданні транзакцій.

```
// List pre-funded account
eth.accounts

// Check balance (dev mode pre-funds with max Ether)
web3.fromWei(eth.getBalance(eth.accounts[0]), "ether")

// Current block number
eth.blockNumber
```

```

Fatal: Bad developer-mode genesis configuration: difficulty must be 0
grusevd@grusevdDavid:~/eth-private$ eth.accounts
eth.accounts: command not found
grusevd@grusevdDavid:~/eth-private$ nano genesis.json
grusevd@grusevdDavid:~/eth-private$ mkdir -p ~/eth-private/devnode
grusevd@grusevdDavid:~/eth-private$ geth --datadir ~/eth-private/devnode \
>   --dev \
>   --http \
>   --http.port 8545 \
>   --http.api eth,net,web3,admin,debug \
>   --http.corsdomain "*" \
>   console 2>> ~/eth-private/logs/node1.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1
at block: 0 (Thu Jan 01 1970 00:00:00 GMT+0000 (UTC))
datadir: /home/grusevd/eth-private/devnode
modules: admin:1.0 debug:1.0 dev:1.0 eth:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> eth.accounts
[{"0x71562b71999873db5b286df957af199ec94617f7"}]
> web3.fromWei(eth.getBalance(eth.accounts[0]), "ether")
1.15792089237316195423570985008687907853269984665640564039457584007913129639927e+59
> eth.blockNumber
0
>

```

Рис. 11: Initialization and pre-funded account details

Крок 8: Створюємо другий рахунок для тестування транзакцій.

```

geth --datadir ~/eth-private/devnode account new
# Enter simple password (e.g., testpass123)

```

Повернемось до консолі Geth (перший термінал) і переконаймося, що обидва облікові записи відображаються:

```
eth.accounts
```

```

grusevd@grusevdDavid:~$ geth --datadir ~/eth-private/devnode account new
INFO [02-18|14:42:00.352] Maximum peer count           ETH=50 total=50
INFO [02-18|14:42:00.354] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or directory"
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key: 0x8A43fCb8a60414Ec2436411a5d0a28c847829969
Path of the secret key file: /home/grusevd/eth-private/devnode/keystore/UTC--2026-02-18T14-42-54.017916010Z--8a43fc8a60414ec2436411a5d0a28c847829969

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!

grusevd@grusevdDavid:~$
```

Рис. 12: Account creation output showing the new address and keystore file path

Крок 9: Створимо першу транзакцію.

Переведемо трохи ефіру з "багатого" dev рахунку на новостворений рахунок і подивимося, чи блок створюється автоматично та подивимося деталі транзакції.

```
// Send 5 Ether from dev account to second account
eth.sendTransaction({
  from: eth.accounts[0],
  to: eth.accounts[1],
  value: web3.toWei(5, "ether")
})

// Dev mode auto-seals a block immediately
eth.blockNumber

// Verify recipient balance
web3.fromWei(eth.getBalance(eth.accounts[1]), "ether")
```

```
> eth.sendTransaction({from: eth.accounts[0], to: eth.accounts[1], value: web3.toWei(5, "ether")})
"0xfeae95df7306d66e25e7039ac18bcd84123118f72e652466bdb82b29c0f5289"
> eth.blockNumber
1
> web3.fromWei(eth.getBalance(eth.accounts[1]), "ether")
5
> eth.getTransaction("0xfeae95df7306d66e25e7039ac18bcd84123118f72e652466bdb82b29c0f5289")
{
  accessList: [],
  blockHash: "0xf08fa8eb647d8a8a62fe28db98a4e05dc3911cb7b5f49c71aacd262427adbb9b",
  blockNumber: 1,
  chainId: "0x539",
  from: "0x71562b71999873db5b286df957af199ec94617f7",
  gas: 21000,
  gasPrice: 875000001,
  hash: "0xfeae95df7306d66e25e7039ac18bcd84123118f72e652466bdb82b29c0f5289",
  input: "0x",
  maxFeePerGas: 2000000001,
  maxPriorityFeePerGas: 1,
  nonce: 0,
  r: "0xc71b98a8fd9f67aaa8e37cf08a5a9057966846af0928911f3d4c9f47f0f74125",
  s: "0x763ecf867f27cb14856e00c90737d21939f42763ae3ac3c43157e4010e5e6fc3",
  to: "0x8a43fc8a60414ec2436411a5d0a28c847829969",
  transactionIndex: 0,
  type: "0x2",
  v: "0x1",
  value: 50000000000000000000000000000000,
  yParity: "0x1"
}
```

Рис. 13: Transaction details and recipient balance

Крок 10: Подивимося деталі блоку та самої транзакції:

```
// Full block information
eth.getBlock(1)

// Transaction receipt (gas used, status, etc.)
eth.getTransactionReceipt("0xTX_HASH_HERE")
```

Оскільки у нас тестова мережа, то маємо автоматичне підтвердження блоків у момент відправлення транзакції – у реальній мережі нам довелося б чекати, поки валідатор включить її (± 12 блоків, або 2-3 хв).

Рис. 14: Block transaction details and metadata

Тут вже можна поглянути на деталі транзакції і самого блоку, куди вона була включена.

- **difficulty**: 0 – підтверджує, що це post-Merge PoS chain;
 - **gasUsed**: 21000 – це стандартна вартість простого переказу ETH. Кожна транзакція Ethereum має базову вартість 21 000 gas. Взаємодії зі смарт-контрактами коштують дорожче, оскільки EVM повинен виконувати код на додаток до цієї базової вартості.
 - **status**: "0x1" – означає, що транзакція була успішною (0x0 означало б невдачу).
 - **miner** – показує на мій dev запис розробника. У PoS це фактично "одержувач комісії", який збирає чайові.

5.5 Спроба 3: Розгортання двох нод у приватній мережі для демонстрації P2P мережі

Відкриємо Geth на другому терміналі (де ми створювали другий акаунт) і створимо другу ноду для peer з'єднання. Це демонструватиме peer-to-peer networking з нашого, описаного в пункті 4, прикладі.

```
grusev@grusev-david:~$ geth --datadir ~/eth-private/node2 \
>   --dev \
>   --http \
>   --http.port 8546 \
>   --port 30304 \
>   --authrpc.port 8552 \
>   console 2>> ~/eth-private/logs/node2.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1
at block: 0 (Thu Jan 01 1970 00:00:00 GMT+0000 (UTC))
datadir: /home/grusev/eth-private/node2
modules: admin:1.0 debug:1.0 dev:1.0 eth:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> admin.nodeInfo.enode
enode://241cf4995f2a0f8a82a997b9d06b5ee9e48ecc4912c6030dd73aaafa215fdc7579e46a114935606f436f2bf34a3709ec9727c6ccb81cf5
0db5e047555b08b72@127.0.0.1:1073
> -
```

Рис. 15: Second node initialization with the same genesis file

Адреса enode закінчується на :0 замість :30303, а це означає, що Node 1 не шейрить свій порт прослуховування правильно, оскільки режим –dev все ж розроблений як одновузлове середовище. І як я одразу про це не прочитав? Ну ладно, доведеться закривати ноду 1 і спробувати перестворити, використавши –dev admin API mode для ручного підключення вузлів. Порт показує :0 у рядку enode, але вузли все одно можуть підключатися, якщо ми використовуємо шлях IPC (Inter-Process Communication) безпосередньо. Спершу перевіримо фактичний стан прослуховування мережі:

```
> net.listening
true
> net.peerCount
0
> admin.nodeInfo.ports
{
  discovery: 0,
  listener: 0
}
>
```

Рис. 16: Network listening and peer count status

`net.listening` є істинним, але обидва порти мають значення 0. Це підтверджує, що режим `–dev` приймає ідею мережевого зв’язку, але фактично не відкриває жодних портів для з’єднання. Це можна назвати дверима з написом «відкрито», але без ручки, ха. Ну що ж, `dev` не працює, то доведеться робити кастомну приватну мережу з двома відповідними вузлами, щоб продемонструвати P2P-мережі та синхронізацію блоків.

Крок 11: Перестворимо структуру genesis block для приватної мережі і ініціалізуємо обидві ноди:

Перестворимо genesis block, задавши високий поріг `terminalTotalDifficulty`, який задовільняє вимогу Geth 1.12.x після злиття, зберігаючи ланцюг у стані, схожому на стан до злиття. В новіших версіях (після 1.12.x) поле `terminalTotalDifficulty` є невід’ємною складовою генезис-блоку. Розробники повністю відмовилися у підтримці ланцюгів, що працюють на PoW і було зроблено параметр `terminalTotalDifficulty` (TTD) критично важливим для ініціалізації або оновлення ланцюгів та перейшли на протокол Proof-of-Stake (PoS).

```
cat > ~/eth-private/genesis-pow.json << 'EOF'
{
  "config": {
    "chainId": 54321,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "berlinBlock": 0,
    "londonBlock": 0,
    "terminalTotalDifficulty": 999999999999
  },
  "alloc": {},
  "coinbase": "0x000000000000000000000000000000000000000000000000000000000000000",
  "difficulty": "0x400",
  "extraData": "0x00",
  "gasLimit": "0x8000000"
```

Значення `999999999999 terminalTotalDifficulty` ланцюг ніколи не досягне шляхом майнінгу, ефективно утримуватиме його в режимі до злиття, одночасно задовольняючи обов'язкові вимоги до конфігурації Geth.

Рис. 17: Contents of updatedgenesis-pow.json

```
geth --datadir ~/eth-private/pow-node1 \
init ~/eth-private/genesis-pow.json
geth --datadir ~/eth-private/pow-node2 \
init ~/eth-private/genesis-pow.json
```

```

grusevd@grusevdDavid:~/eth-private$ geth --datadir ~/eth-private/pow-node1 \
>   --networkid 54321 \
>   --nodiscover \
>   --nat none \
>   --port 30303 \
>   --http \
>   --http.port 8545 \
>   --http.api eth,net,web3,admin,miner \
>   --allow-insecure-unlock \
>   console 2>> ~/eth-private/logs/pow-node1.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1
at block: 0 (Thu Jan 01 1970 00:00:00 GMT+0000 (UTC))
datadir: /home/grusevd/eth-private/pow-node1
modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> admin.nodeInfo.enode
"enode://04b6a212d6ec442421f4aa7c826fdb04fee621c31dd51392cd59a9719d5af258568095e6f174cc727c83
67e06e30dcba1449dba105139faed88836c50dbe80@127.0.0.1:30303?discport=0"
> admin.nodeInfo.ports
{
  discovery: 0,
  listener: 30303
}

grusevd@grusevdDavid:~$ geth --datadir ~/eth-private/pow-node2 \
--networkid 5 >   --networkid 54321 \
--no>   --nodiscover \
>   --nat none \
>   --port 30304 \
>   --http \
>   --http.port 8546 \
>   --http.api eth,net,web3,admin,miner \
--aut>   --authrpc.port 8552 \
>   --allow-insecure-unlock \
  console 2>> ~/eth-private/logs/pow-node2.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1
at block: 0 (Thu Jan 01 1970 00:00:00 GMT+0000 (UTC))
datadir: /home/grusevd/eth-private/pow-node2
modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> admin.nodeInfo.enode
"enode://29aedfe5973fb3206452dfea650b00fe6872cb5b2db080273f75fe9b541c69a5f28bd10bb0b431d66140b
4c64cff301bd34ff2a76693b7ae4ccfb99516aede03@127.0.0.1:30304?discport=0"
> admin.nodeInfo.ports
{
  discovery: 0,
  listener: 30304
}

```

Рис. 18: Initialization output from both nodes with matching genesis hashes

Обидва вузли ініціалізовані однаковим гешем генезис блоку. Це є надзвичайно важливим, оскільки вузли можуть з'єднуватися лише за умови, що вони мають одинаковий блок генезису.

Крок 12: Підключаємо вузли один до одного для формування приватної мережі та перевіряємо зв'язок.

В терміналі 1 (консолі Node 1), з'ясовуємо enode URL:

```
admin.nodeInfo.enode
```

Копіюємо цей рядок enode. У терміналі 2 (консолі Node 2) додаємо Node 1 як рівноправного партнера:

```
admin.addPeer("enode://04b6a2...@127.0.0.1:30303")

// Verify connection
net.peerCount
```

admin.peers

The screenshot shows two terminal windows side-by-side. Both windows have a purple title bar with the text 'grusevd@grusevdDavid: ~'. The left window has a white background and displays the command 'admin.addPeer("enode://...")' followed by its output. The right window has a black background and displays the command 'net.peerCount' followed by its output.

```
grusevd@grusevdDavid: ~
> admin.addPeer("enode://04b6a212d6ec442421f4aa7c826fdb04fee621c31dd51392cd59a9719d5af2585680 ^95e6f174cc727c8367e06e30dcba105139faed88836c50dbe80@127.0.0.1:30303")
true
> net.peerCount
1
> admin.peers
[{
  caps: ["eth/68", "eth/69", "snap/1"],
  enode: "enode://04b6a212d6ec442421f4aa7c826fdb04fee621c31dd51392cd59a9719d5af258568095e6f174cc727c8367e06e30dcba105139faed88836c50dbe80@127.0.0.1:30303",
  id: "bdb2fabb75398cac938a80f924a91ccdf98a3023e69a7209e0a46776fb70a3e3",
  name: "Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1",
  network: {
    inbound: false,
    localAddress: "127.0.0.1:48926",
    remoteAddress: "127.0.0.1:30303",
    static: true,
    trusted: false
  },
  protocols: {
    eth: {
      earliestBlock: 0,
      latestBlock: 0,
      latestBlockHash: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
      version: 69
    },
    snap: {
      version: 1
    }
  }
}
grusevd@grusevdDavid: ~/eth-private
> net.peerCount
1
> admin.peers
[{
  caps: ["eth/68", "eth/69", "snap/1"],
  enode: "enode://29aedfe5973fb3206452dfa650b00fe6872cb5b2db000273f75fe9b541c69a5f28bd10bb0b431d66140b4c64cff301bd34ff2a76693b7ae4ccfb99516ae03@127.0.0.1:48926",
  id: "598b84af91e00f93cd503e7be4597e8b3f31dc7caaea860022e4d3cf13584e26",
  name: "Geth/v1.16.9-stable-95665d57/linux-amd64/go1.25.1",
  network: {
    inbound: true,
    localAddress: "127.0.0.1:30303",
    remoteAddress: "127.0.0.1:48926",
    static: false,
    trusted: false
  },
  protocols: {
    eth: {
      earliestBlock: 0,
      latestBlock: 0,
      latestBlockHash: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
      version: 69
    },
    snap: {
      version: 1
    }
  }
}]
```

Рис. 19: Success peer connection of Node 1 to Node 2's

Вихідні дані `admin.peers` містять важливу інформацію про нашу мережу:

- `caps: ["eth/68" "eth/69" "snap/1"]` — підтримувані версії протоколу.
- `inbound: false` на Node 2 означає, що він ініціював з'єднання; Node 1 показує `inbound: true` (отримав з'єднання).
- Обидва вузли комунікують за допомогою протоколу Ethereum wire (`eth`) та протоколу синхронізації знімків (`snap`).

В вересні 2022 року з переходом PoW на PoS процес майнінгу (mining API) ефіру (ETH) було припинено. Тому, на жаль, не вийде показати цей процес з подальшою синхронізацією нового блоку на всі ноди навіть у тестовій мережі.

```

> miner.setEtherbase(eth.accounts[0])
TypeError: Object has no member 'setEtherbase'
at <eval>:1:19(6)

> miner
{
  setExtra: function(),
  setGasLimit: function(),
  setGasPrice: function()
}

```

Рис. 20: Attempting to use deprecated mining commands in Geth console

Крок 13: Проаналізуємо хоч конфігурацію та статус вузлів у мережі.

```

// Full node information
admin.nodeInfo

// Protocol configuration
admin.nodeInfo.protocols
// Shows chainId, terminalTotalDifficulty, genesis hash

// Network ID
net.version

// Listening status and ports
net.listening
admin.nodeInfo.ports

```

```

{
  admin.nodeInfo
  {
    enode: "0x449db1e95139faed88816c50fbbe80f127.0.0:30303?discovery=0",
    enr: "enr://y4QmRrgbjGmVtC2d0KyrbnyE6EQk32mAg7b2-xRtjsFnF0u7X_tgobooH3nVQ8mJ9PAUF-Hld_NutryouilsmI0GAZxx5he_g
    vNvHn0hds55Aglkgm@0gm1uhH3dAM5z2VjD01NmsoQlEtq1sIxEXJChBqnyC9ugt-51HDHdUTks12qXGdWV3YRzbmFWiN8y3Cc018",
    id: "54321",
    ip: "127.0.0.1",
    listenAddr: "[::]:30303",
    name: "Geth/v1.10.9-stable-95665d57/linux-amd64/go1.25.1",
    ports: {
      discovery: 0,
      listener: 30303
    },
    protocols: {
      eth: {
        config: {
          berlinBlock: 0,
          byzantiumBlock: 0,
          chainid: 54321,
          constantinopleBlock: 0,
          depositContractAddress: "0x0000000000000000000000000000000000000000000000000000000000000000",
          eip150Block: 0,
          eip158Block: 0,
          homesteadBlock: 0,
          istanbulBlock: 0,
          londonBlock: 0,
          petersburgBlock: 0,
          terminalTotalDifficulty: 099999999999
        },
        genesis: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
        head: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
        network: 54321
      },
      snap: {}
    }
  }
}

```

Рис. 21: admin.nodeInfo with node configuration and active protocols

```

> admin.nodeInfo.protocols
> admin.nodeInfo.protocols.eth.config
eth: {
  config: {
    berlinBlock: 0,
    byzantiumBlock: 0,
    chainId: 54321,
    constantinopleBlock: 0,
    depositContractAddress: "0x0000000000000000000000000000000000000000000000000000000000000000",
    eip150Block: 0,
    eip158Block: 0,
    eip158Block: 0,
    homesteadBlock: 0,
    istanbulBlock: 0,
    londonBlock: 0,
    petersburgBlock: 0,
    terminalTotalDifficulty: 099999999999
  },
  genesis: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
  head: "0x52645e65a985a2145d8bf596ac493c09e41e2985268b1df43406d03d3869b5f4",
  network: 54321
},
snap: {}
> net.version
"54321"
> net.listening
true
>

```

Рис. 22: admin.nodeInfo.protocols with chain configuration and active EIPs

Розділ `admin.nodeInfo.protocols.eth.config` відображає повну конфігурацію ланцюга, включаючи всі активовані EIP. Ці дані підтверджують, що обидва вузли працюють з однаковою конфігурацією приватної мережі та успішно спілкуються один з одним.

5.6 Проблеми та їх рішення

У процес розгортання блокчейну, я виявив кілька істотних відмінностей між інструкціями знайденими в онлайн-підручниках (більшість з яких написані для Geth 1.9–1.13, 2019–20 р.) та реальністю роботи

Geth 1.16.9. Проблеми та їхні рішення зібрали у табличку.

№	Problem	Cause	Solution
1	Fatal: 'terminalTotalDifficulty' не встановлено в genesis-блоці	Geth 1.13.x потребує конфігурації після Merge; файли старого зразка без цього поля відхиляються	Додали 'terminalTotalDifficulty' до конфігурації genesis
2	<code>miner.setEtherbase()</code> та <code>miner.start()</code> відсутні	Geth 1.16.x видалив команди PoW-майнінгу після переходу Ethereum на PoS	Використав режим <code>--dev</code> для транзакцій (автоматичне запечатування блоків)
3	Dev-режим повідомляє про P2P нульові порти і не дає з'єднатися	Режим <code>--dev</code> навмисно вимикає P2P мережу	Використав окрему кастомну мережу для демонстрації P2P
4	Genesis без 'terminalTotalDifficulty' відхиляється навіть для PoW ланцюгів	Geth 1.16.x вимагає конфігурацію post-Merge для всіх ланцюгів	Встановив 'terminalTotalDifficulty' на недосяжно високе значення (999999999999)

Таблиця 6: Проблеми, що виникли під час розгортання, та їх вирішення

Ці питання ілюструють важливу практичну реальність: переход Ethereum від Proof-of-Work до Proof-of-Stake (в оновленні «The Merge», вересень 2022 року) кардинально змінив робочий процес розгортання. Більшість онлайн-посібників були написані для версій Geth до Merge і зараз частково або повністю застаріли/змінилися (конкретні команди, параметри конфігурації), але основні концепції залишаються актуальними (блоки генезису, виявлення однорангових вузлів, URL-адреси enode, консоль JavaScript).

6 Висновки

Провівши порівняльний аналіз п'яти основних блокчейн-систем, таких як Ethereum, Bitcoin, Dash, NEO та Litecoin можу підбити такі важливі висновки щодо розгортання та взаємозамінності модулів:

- Архітектура визначає сумісність.** Системи, що мають спільного предка по кодовій базі (Bitcoin → Litecoin, Bitcoin → Dash), демонструють найвищий ступінь структурної схожості та потенціал для повторного використання модулів. Ethereum і NEO, незважаючи на те, що обидві підтримують смарт-контракти, мають фундаментально несумісні архітектури.
- Консенсус є найменш портативним модулем.** Механізм консенсусу кожної системи тісно пов'язаний з її форматом блоків, логікою переходу між станами та мережевим протоколом. Заміна одного механізму консенсусу на інший вимагає фактично перепроектування всієї системи.
- Сховище є найбільш портативним модулем.** Використання загальних сховищ ключ-значення (LevelDB, RocksDB) у всіх системах означає, що бекенд сховища можна замінити з мінімальним впливом на поведінку протоколу.
- Модель стану створює фундаментальний розрив.** Account-based системи (Ethereum, NEO) та системи на основі UTXO (Bitcoin, Litecoin, Dash) обробляють транзакції принципово різними способами, що робить сумісність модулів між моделями непрактичним.
- Приватні мережі Ethereum є найбільш практичним вибором** для освітнього впровадження, пропонуючи багату функціональність (рахунки, транзакції, майнінг(в минулому), смарт-контракти, однорангові мережі) з помірними вимогами до апаратного забезпечення.

References

- [1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. 2008. ([URL](#)).
- [2] Vitalik Buterin. «Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform». In: *White Paper* (2014). ([URL](#)).
- [3] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper)*. Tech. rep. 2018. ([URL](#)).
- [4] Evan Duffield and Daniel Diaz. *Dash: A Payments-Focused Cryptocurrency*. Tech. rep. 2015. ([URL](#)).
- [5] NEO Foundation. *NEO White Paper: A Distributed Network for the Smart Economy*. Tech. rep. NEO Foundation, 2017. ([URL](#)).
- [6] Charlie Lee. *Litecoin — Open Source P2P Digital Currency*. 2011. ([URL](#)).
- [7] Nalaka Pathirana. *How to set up a Private Ethereum Blockchain*. 2019. ([URL](#)).
- [8] Pradeep Thomas. *How to setup your own private Ethereum network*. 2020. ([URL](#)).
- [9] Guy Dagan. *The Actual Networking behind the Ethereum Network*. 2018. ([URL](#)).