

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»

КАФЕДРА _____

СПЕЦІАЛЬНІСТЬ _____

(шифр та назва)

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА _____

(назва)

**ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА
РОБОТА**

на тему:

**Розробка програми, яка вимірює швидкість завантаження і вивантаження даних
між клієнтом і сервером через мережу. Програма має підтримувати протоколи
передачі даних TCP та UDP**

(назва затвердженої теми)

(на прикладі _____)

Студент (ка) _____ Тимофєєва Анастасія Дмитрівна _____
(підпис студента) (прізвище, ім'я по-батькові студента)

Науковий
керівник _____
(підпис наукового керівника) (прізвище, ініціали, наукове звання і ступінь)

Допущено до захисту “_____” 202_ р.

Завідувач кафедри _____
(підпис) (прізвище, ініціали, наукове звання і ступінь)

Декан факультету _____
(підпис) (прізвище, ініціали)

Завідувач
навчальної частини _____
(підпис) (прізвище, ініціали)

Київ – 2025

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»

КАФЕДРА _____

СПЕЦІАЛЬНІСТЬ _____

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА _____

ЗАВДАННЯ

на випускню кваліфікаційну бакалаврську роботу студентів (ці):

(прізвище, ім'я, по-батькові студента (ки))

1. Тема роботи _____

2. Затверджена наказом по університету від “ _____ ” 202 _____ р. № _____

3. Цільова установка на вихідні дані до роботи

4. Перелік графічного матеріалу (з точним визначенням обов'язкових креслень) _____

5. Консультанти по роботі, із зазначенням розділів, що їх стосуються:

Розділ	Науковий керівник	Підпис, дата	
		Завдання видав	Завдання отримав

6. Зміст випускної кваліфікаційної бакалаврської роботи (перелік питань)

7. Календарний план виконання випускної кваліфікаційної бакалаврської роботи

№ з/п	Назва етапів випускної кваліфікаційної бакалаврської роботи	Термін виконання етапів випускної кваліфікаційної бакалаврської роботи	
		За планом	Фактично
1.	Обрання теми випускної кваліфікаційної магістерської роботи (ВКБР)	05.08.24 – 13.09.24	03.10.2024
2.	Опрацювання літератури та інших джерел для написання ВКБР	16.09.24 – 01.11.24	21.10.2024
3.	Виконання 1 розділу ВКБР	04.11.24 – 28.02.25	26.02.2025
4.	Виконання 2 розділу ВКБР	03.03.25 – 21.03.25	18.03.2025
5.	Виконання 3 розділу ВКБР, висновки	24.03.25 – 18.04.25	16.04.2025
6.	Передзахист та проведення перевірки на плагіат	21.04.24 – 02.05.25	30.04.2025
7.	Відгук керівника, рецензія	01.05.25 – 12.05.25	08.05.2025

8. Дата видачі завдання “_____” 202_ р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

Анотація

У дипломній роботі представлено розробку клієнт-серверного програмного забезпечення для вимірювання швидкості передачі даних у мережі з використанням транспортних протоколів TCP та UDP. Актуальність дослідження обумовлена зростаючою потребою у точному аналізі продуктивності мережових з'єднань у контексті поширення потокового відео, VoIP-зв'язку, хмарних сервісів і високонавантажених онлайн-платформ. Розроблений інструмент дозволяє вимірювати реальну пропускну здатність каналу зв'язку, оцінювати втрати пакетів, варіабельність затримки (jitter) та стабільність з'єднання в режимі реального часу.

У ході виконання роботи проаналізовано існуючі підходи до вимірювання мережової швидкості, зокрема програмні засоби Speedtest, iPerf, NetSpeedMonitor. Здійснено дослідження принципів функціонування протоколів TCP та UDP, визначено їхні особливості, переваги та обмеження для різних типів трафіку. Обґрунтовано вибір інструментів реалізації: мови програмування Python, бібліотек socket, threading, tkinter.

Реалізовано архітектуру клієнт-серверної взаємодії, де клієнт генерує потік даних і передає їх на сервер, що фіксує обсяг прийнятої інформації та обчислює середню швидкість. Передбачено підтримку обох протоколів — TCP для надійної доставки і UDP для швидкісного аналізу з урахуванням втрат. Графічний інтерфейс забезпечує зручне керування процесом тестування, виведення результатів та повідомлень.

Проведено серію експериментальних досліджень у різних умовах: локально (localhost), у локальній мережі (LAN), через Wi-Fi-з'єднання, а також при створенні штучного навантаження. Отримані результати підтвердили стабільність роботи, точність обчислень та відповідність практичним очікуванням. Виконано порівняльний аналіз ефективності TCP та UDP,

визначено доцільність їх використання для різних задач — від завантаження файлів до реального часу.

Узагальнено практичні рекомендації щодо застосування програми для побутового використання, адміністрування мереж, навчального процесу та технічної підтримки. Окреслено перспективні напрями подальшого розвитку: реалізація багатокористувацького режиму, візуалізації результатів, кросплатформенності, мобільних версій та інтеграції з системами моніторингу.

Результати роботи мають як наукову, так і прикладну цінність, а створене програмне забезпечення є ефективним інструментом для аналізу та діагностики мережевих з'єднань у сучасних інформаційних системах.

Abstract

This thesis presents the development of a client-server software solution for measuring data transfer speed over a network using TCP and UDP transport protocols. The relevance of the research is driven by the increasing demand for accurate network performance analysis due to the widespread use of streaming media, VoIP communication, cloud services, and high-load online platforms. The developed tool enables real-time measurement of actual channel throughput, packet loss, jitter, and connection stability.

During the course of the work, existing methods for network speed measurement were analyzed, including software tools such as Speedtest, iPerf, and NetSpeedMonitor. The principles of TCP and UDP protocol operation were studied, with a focus on their characteristics, advantages, and limitations depending on traffic type. Python was chosen as the development language, utilizing libraries such as socket, threading, and tkinter for low-level networking and graphical interface implementation.

The system architecture is based on client-server interaction, where the client sends a continuous data stream to the server. The server records the amount of received data and calculates the average transfer speed. Both protocols are supported: TCP provides reliable delivery, while UDP allows for high-speed testing with packet loss analysis. The user interface enables easy test control and displays results and diagnostic messages.

A series of experimental tests were conducted under various conditions: on a single device (localhost), over a local area network (LAN), via Wi-Fi, and under artificial load. The results confirmed the stability of the application, accuracy of calculations, and practical applicability. A comparative analysis of TCP and UDP was performed, highlighting the appropriateness of each protocol for different scenarios — from file transfers to real-time streaming.

Practical recommendations for using the software were formulated for home users, network administrators, educational purposes, and technical support. Potential directions for further development were outlined, including multi-client support, graphical visualization of results, cross-platform compatibility, mobile versions, and integration with monitoring systems.

The outcomes of this research have both scientific and practical value. The developed software represents an effective tool for analyzing and diagnosing network connections in modern information systems.

ЗМІСТ

1. ВСТУП	1
1.1. Актуальність теми	1
1.2. Мета і завдання дослідження	3
1.3. Об'єкт і предмет дослідження	4
1.4. Практичне значення одержаних результатів	5
1.5. Апробація результатів та публікації	6
2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	9
2.1. Аналіз існуючих методів вимірювання швидкості передачі	9
Вимірювання часу відгуку (ping, RTT)	10
Завантаження / вивантаження тестових файлів	10
Аналіз пакетової передачі	11
Синтетичні тести (Speedtest, iPerf)	11
2.2. Огляд існуючих програмних рішень	12
2.3. Порівняльний аналіз протоколів TCP та UDP	14
2.4. Мережеві параметри, що впливають на швидкість	15
3. ВИБІР АРХІТЕКТУРИ ТА ІНСТРУМЕНТІВ РОЗРОБКИ	18
3.1. Вибір технологій та середовища розробки	19
3.2. Опис архітектури клієнт-серверної системи	20
3.3. Алгоритми вимірювання швидкості	22
3.4. Схема взаємодії клієнта і сервера	23
4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	24
4.1. Серверна частина	25
4.1.1. Структура серверного додатка	25
4.1.2. Обробка запитів та відповіді сервера	26

4.1.3. Підтримка TCP та UDP	28
4.2. Клієнтська частина	30
4.2.1. Інтерфейс користувача	30
4.2.2. Взаємодія з сервером через TCP/UDP	32
4.2.3. Візуалізація результатів	35
4.3. Формати обміну даними	37
4.4. Обробка помилок і збоїв	39
5. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ	41
5.1. Методика тестування	41
5.2. Аналіз отриманих результатів	49
5.3. Порівняння TCP і UDP	54
6. ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	66
ДОДАТКИ	70

ВСТУП

1.1. Актуальність теми

Сучасний розвиток інформаційних технологій зумовлює необхідність ефективних методів оцінки продуктивності мережевих з'єднань. Вимірювання швидкості завантаження та вивантаження даних між клієнтом і сервером є важливим завданням не лише для адміністраторів мереж, а й для кінцевих користувачів. Протоколи TCP та UDP мають різні характеристики, які впливають на швидкість передачі даних, тому їх порівняльний аналіз дозволяє визначити оптимальні умови використання кожного з них. У сучасному цифровому середовищі, де обсяги переданої інформації постійно зростають, ці аспекти стають особливо важливими для забезпечення високої якості обслуговування.

У сучасному світі, де цифрові технології проникають у всі сфери життя, мережеві з'єднання відіграють критично важливу роль у забезпеченні безперебійної роботи різноманітних сервісів. Від високошвидкісного з'єднання залежить не лише ефективність роботи бізнесу, а й комфорт користувачів при взаємодії з онлайн-ресурсами. Використовувані протоколи значно впливають на швидкість і надійність з'єднань, тому їх детальний аналіз і порівняння є важливими для оптимізації роботи мережі. Врахування характеристик різних протоколів та їх правильне налаштування допомагають досягти найкращих результатів у швидкості, стабільності та безпеці мережевого з'єднання.

В умовах цифрової трансформації та активного розвитку інтернет-технологій якість мережевого з'єднання має ключове значення для багатьох сфер діяльності. Популярність потокового відео, хмарних сервісів, дистанційного навчання та онлайн-ігор підвищує вимоги до стабільності та швидкості зв'язку. Відсутність ефективних методів вимірювання продуктивності мережі може спричинити зниження якості обслуговування, збільшення затримок та нестабільну роботу сервісів, що негативно позначається

як на користувацькому досвіді, так і на репутації компаній, що надають ці послуги.

Зростання обсягу даних та необхідність обслуговування великої кількості користувачів одночасно ставлять нові вимоги до мережевої інфраструктури. Особливо важливими стають такі параметри, як надійність з'єднання, якість обслуговування (QoS) та здатність швидко адаптуватися до змін у навантаженні. Підвищені потреби у пропускній здатності каналів змушують операторів та адміністраторів шукати нові рішення для ефективного використання ресурсів та вдосконалення технологій. Фактори, такі як мінімізація затримок, зменшення втрат пакетів та максимальна швидкість передачі даних, є ключовими критеріями для забезпечення безперебійної роботи та високої якості послуг.

На швидкість передачі даних впливає багато факторів, зокрема пропускна здатність каналу, рівень втрат пакетів, налаштування мережевого обладнання та загальне навантаження на мережу. Однак важливо не лише вимірювати ці параметри, а й відстежувати їх зміни у реальному часі, що дозволяє швидко адаптувати мережу до нових умов. Крім того, необхідно враховувати взаємодію між різними рівнями мережевої інфраструктури, оскільки навіть незначні зміни на одному з етапів передачі даних можуть суттєво вплинути на загальну продуктивність. Аналіз цих факторів і розробка відповідного програмного забезпечення для оцінки продуктивності дозволять підвищити ефективність роботи мережевих сервісів і покращити користувацький досвід.

Крім того, подібні дослідження допоможуть виявити вразливості мережевих з'єднань та підвищити їхню безпеку, що є важливою частиною сучасних вимог до інформаційних технологій. Для досягнення цієї мети необхідно розробити нові методи та інструменти для точного вимірювання продуктивності та стабільності мережі. Використання сучасних алгоритмів для аналізу трафіку, прогнозування навантаження та адаптивного управління пропускною здатністю є необхідною умовою для забезпечення високих стандартів якості на всіх етапах обробки даних.

Таким чином, ефективне впровадження та використання цих методів не лише дозволить оптимізувати роботу мережевої інфраструктури, а й сприятиме розвитку нових сервісів і послуг, які забезпечать ще більшу зручність і доступність для користувачів у умовах зростаючого навантаження на мережі.

1.2. Мета і завдання дослідження

Мета:

Розробка програмного засобу для вимірювання швидкості передачі даних у мережі з підтримкою протоколів TCP та UDP.

Завдання:

- Аналіз існуючих методів вимірювання швидкості передачі даних.
- Дослідження принципів роботи протоколів TCP та UDP.
- Вибір відповідних технологій та середовища розробки для створення програмного засобу.
- Реалізація клієнт-серверного застосунку для вимірювання швидкості передачі даних.
- Проведення експериментальних тестувань із використанням розробленого застосунку.
- Аналіз отриманих результатів, визначення переваг та недоліків TCP і UDP у контексті передачі даних.
- Формулювання рекомендацій щодо вибору оптимального протоколу залежно від характеристик мережевого середовища.

Окрім основних завдань, дослідження охоплює аналіз факторів, що впливають на швидкість передачі даних, таких як конфігурація мережевого

обладнання, тип підключення та поточне мережеве навантаження. Отримані результати допоможуть розробити рекомендації з оптимізації роботи мережевих з'єднань. Зокрема, важливими аспектами будуть: інтерференція між пакетами, управління потоком, механізми контролю помилок та динамічна адаптація до змінних умов мережі.

Дослідження також передбачає аналіз роботи протоколів у різних умовах навантаження. Для TCP будуть розглянуті такі аспекти, як механізми відновлення пакетів, управління перевантаженнями мережі та вплив затримок на ефективність передачі даних. Для UDP основна увага приділятиметься наслідкам відсутності механізмів підтвердження доставки, що може бути критично важливим у ситуаціях, де першочергове значення має швидкість передачі, а не гарантія отримання кожного пакета.

Окремо буде досліджено вплив різних типів мережевих з'єднань (кабельних, бездротових, оптоволоконних) на ефективність передачі даних через TCP та UDP. Різні типи підключень можуть мати суттєвий вплив на швидкість і стабільність з'єднань, що необхідно враховувати при оцінці ефективності використання тих чи інших мережевих протоколів у конкретних умовах. Важливо також визначити параметри, які забезпечують максимальну продуктивність залежно від типу мережі та її поточного стану.

Крім того, у межах дослідження буде проведене порівняльне тестування різних варіантів налаштувань програмного засобу для забезпечення максимально ефективного використання пропускну здатності мережевого каналу та мінімізації затримок. Це дозволить створити гнучкий інструмент, який можна адаптувати до конкретних потреб користувача або вимог мережевого середовища.

На основі отриманих даних буде сформульовано низку практичних рекомендацій для адміністраторів мереж, а також для кінцевих користувачів, які прагнуть покращити якість та швидкість своїх мережевих з'єднань. Ці

рекомендації охоплюватимуть вибір оптимального протоколу, налаштування мережевого обладнання та визначення найкращих параметрів для забезпечення максимальної ефективності роботи мережі в умовах різного навантаження.

1.3. Об'єкт і предмет дослідження

Об'єкт дослідження – процес передачі даних у комп'ютерних мережах.

Предмет дослідження – методи та засоби вимірювання швидкості передачі даних у мережах із використанням протоколів TCP та UDP.

Дослідження спрямоване на оцінку ефективності різних мережевих протоколів, вивчення факторів, що впливають на швидкість передачі даних, а також розробку інструменту, який дозволить вимірювати продуктивність мережі. Передача даних у комп'ютерних мережах є складним процесом, що залежить від багатьох чинників: пропускної здатності каналу, рівня затримок, якості сигналу та ефективності обробки інформації на різних етапах маршрутизації.

Окрему увагу приділено аналізу протоколів TCP та UDP, оскільки вони реалізують принципово різні підходи до обміну даними. TCP гарантує надійність передачі, контроль помилок і підтвердження отримання даних, що робить його оптимальним для застосувань, де важлива точність і цілісність інформації. Натомість UDP працює без встановлення з'єднання, забезпечуючи високу швидкість і мінімальні затримки, що критично важливо для потокового відео, онлайн-ігор та інших сценаріїв, де швидкість переважає над повною достовірністю даних.

Важливо також дослідити, які фактори впливають на ефективність передачі даних. Це можуть бути як фізичні характеристики мережі, так і програмні налаштування. Аналіз таких параметрів, як затримки, втрати пакетів та особливості маршрутизації, допоможе краще зрозуміти продуктивність TCP та UDP, а також визначити оптимальні умови для їх використання.

Окремо розглядається вплив конфігурації мережевого обладнання – маршрутизаторів, комутаторів, серверів – на швидкість і стабільність передачі даних. Також вивчається вплив мережевого навантаження, яке може змінюватися залежно від часу доби, кількості активних користувачів та загального рівня трафіку.

Крім того, дослідження передбачає моніторинг реального мережевого середовища для виявлення можливих проблем, таких як фізичні обмеження каналів зв'язку, збої в синхронізації сигналів або інші перешкоди, що можуть сповільнювати або спотворювати передачу даних.

Практичним результатом цього дослідження стане розробка програмного інструменту для вимірювання швидкості передачі даних. Він дозволить не лише отримати точні показники швидкості в реальних умовах, а й оцінити ефективність TCP та UDP за різних параметрів мережевого середовища. Це допоможе виявити слабкі місця мережевих з'єднань і сформулювати рекомендації для їх оптимізації.

Таким чином, у цьому дослідженні будуть розглянуті не лише теоретичні аспекти функціонування протоколів TCP та UDP, а й їх практичне застосування для створення ефективних інструментів аналізу та вимірювання швидкості передачі даних у реальних умовах експлуатації.

1.4. Практичне значення одержаних результатів

Розроблене програмне забезпечення відкриває нові можливості для адміністраторів мереж і кінцевих користувачів у швидкому та точному вимірюванні швидкості мережевого з'єднання. Це дозволяє визначити, який протокол найкраще підходить для конкретних умов і потреб, що сприяє підвищенню ефективності та стабільності роботи мережі. Користувачі зможуть обирати між різними протоколами для досягнення оптимального балансу між швидкістю передачі даних і надійністю — особливо це важливо при змінних умовах навантаження або специфічних вимогах до трафіку.

Практичне значення цієї розробки полягає також у застосуванні результатів дослідження для моніторингу мережевого трафіку та діагностики несправностей. Завдяки точним вимірюванням, адміністратори зможуть швидше виявляти проблеми з затримками, втратою пакетів або перевантаженням каналів. Це дозволить оперативно реагувати і вжити заходів для поліпшення роботи мережі, що особливо критично для корпоративних мереж і онлайн-сервісів, де навіть незначні збої можуть призвести до серйозних наслідків.

Завдяки інтеграції програмного забезпечення в хмарні сервіси можна автоматично коригувати параметри передачі даних в реальному часі. Це забезпечить динамічне управління мережею, адаптуючись до змінних умов і забезпечуючи оптимальну продуктивність для кожного користувача чи сервісу. Таке автоматичне налаштування дозволяє знижувати навантаження на мережу і підвищувати якість обслуговування, навіть коли мережа перевантажена.

Програмне забезпечення також стане корисним в навчальних процесах, допомагаючи студентам і фахівцям зрозуміти, як працюють різні протоколи передачі даних, а також як мережа поводить себе при різних навантаженнях. Це стане ефективним інструментом для демонстрацій і лабораторних робіт, що дасть студентам практичний досвід роботи з мережами і вивчення принципів адміністрування.

Завдяки простоті у використанні та гнучкості програмний засіб може бути корисним для малого та середнього бізнесу. Він дозволить швидко оцінити якість мережевого з'єднання без потреби у глибоких технічних знаннях. Це стане важливим інструментом для адміністраторів, допомагаючи їм ефективно управляти корпоративними мережами і забезпечувати високу якість обслуговування кінцевих користувачів.

У контексті швидкого зростання обсягів мережевого трафіку та появи нових технологій, результати цього дослідження можуть суттєво сприяти

розвитку нових методів моніторингу і управління мережею, що, в свою чергу, позитивно вплине на стійкість інтернет-інфраструктури та покращить користувацький досвід.

1.5. Апробація результатів та публікації

Результати цього дослідження мають великий потенціал для публікацій у наукових журналах, доповідей на конференціях та активного використання в навчальному процесі. Розроблене програмне забезпечення для вимірювання швидкості передачі даних і аналізу ефективності мережевих протоколів відкриває можливості для глибокого вивчення і порівняння різних мережевих технологій. Це може стати основою для нових наукових статей і досліджень у галузі інформаційних технологій, телекомунікацій і мережевих систем.

Один з ключових аспектів — це вдосконалення методів тестування продуктивності мереж, що стане важливим для телекомунікаційних компаній, розробників програмного забезпечення для мереж та дослідників. Постійний моніторинг і оцінка якості з'єднань для таких компаній є необхідністю, оскільки дозволяє вчасно виявляти проблеми і оптимізувати якість обслуговування кінцевих користувачів. Результати дослідження можуть стати основою для створення нових інструментів, які дозволять ефективно оцінювати продуктивність мереж, враховуючи різні протоколи та умови мережевого середовища.

Що стосується практичних застосувань, то ці результати можуть бути інтегровані у роботу мережевих операційних центрів (NOC), які займаються моніторингом та діагностикою мережевого трафіку. Вони допоможуть підвищити ефективність виявлення проблем і поліпшити автоматизацію процесів, таких як моніторинг продуктивності мереж у реальному часі та оптимізація параметрів підключення. Це дозволить компаніям швидше реагувати на проблеми і покращувати якість обслуговування.

Також, ці результати знайдуть своє застосування в навчальних закладах, де вони допоможуть студентам і викладачам краще зрозуміти теоретичні та практичні аспекти роботи мережевих протоколів. Для курсів з інформаційних технологій і телекомунікацій ці результати стануть чудовою основою для лабораторних робіт, дослідницьких проектів та практичних занять, що дозволяють наочно вивчити різні методи тестування і оптимізації продуктивності мереж.

Крім того, ці результати можуть бути корисними для розробників мережевого обладнання та програмного забезпечення. Наприклад, програми для налаштування мережевого обладнання та моніторингу трафіку можуть використовувати нові методи тестування, що дозволяють точніше оцінювати продуктивність мережі. Це допоможе швидше виявляти можливі проблеми і оптимізувати роботу мереж.

Отже, практичне впровадження результатів дослідження, через публікації та інтеграцію в галузі телекомунікацій та ІТ, сприятиме розвитку новітніх методів управління та оптимізації мереж, а також відкриє нові можливості для моніторингу і підвищення ефективності мережевих з'єднань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1. Аналіз існуючих методів вимірювання швидкості передачі даних

У сучасних комп'ютерних мережах вимірювання швидкості передачі даних є критично важливим завданням для забезпечення стабільної та ефективної роботи сервісів. Для цього використовуються різні методи, кожен з яких має свої переваги та обмеження. Аналіз існуючих методів дозволяє обрати найбільш відповідний підхід для розробки програмного засобу.

Вимірювання часу відгуку (ping, RTT)

Одним із найпростіших методів оцінки продуктивності мережі є вимірювання часу відгуку (Round-Trip Time, RTT) за допомогою утиліти ping. Цей метод передбачає надсилання ICMP-запитів до цільового сервера і вимірювання часу, за який відповідь повертається.

Переваги:

- Простота реалізації та використання;
- Низьке навантаження на мережу;
- Дозволяє оцінити рівень затримки в мережі.

Недоліки:

- Не дає повного уявлення про фактичну швидкість передачі даних;
- ICMP-пакети можуть оброблятися по-різному на різних маршрутизаторах, що впливає на точність результатів.

Використання тестових файлів для завантаження та вивантаження

Цей метод передбачає завантаження або вивантаження файлу певного розміру з тестового сервера та визначення середньої швидкості передачі. Наприклад, такий підхід застосовується в деяких інструментах для вимірювання швидкості інтернет-з'єднання.

Переваги:

- Простий у реалізації;
- Відображає реальні умови передачі даних у мережі;
- Може бути адаптований для різних типів з'єднань.

Недоліки:

- Велика залежність від розміру файлу та обмежень пропускної здатності сервера;
- Низька точність вимірювань у разі нестабільного з'єднання.

Аналіз пакетової передачі (Packet Sniffing)

Цей метод передбачає перехоплення та аналіз мережевого трафіку для оцінки швидкості передачі даних. Використовуються спеціальні інструменти, такі як Wireshark або tcpdump, які дозволяють детально аналізувати пакети, що передаються мережею.

Переваги:

- Надання детальної інформації про мережевий трафік;
- Можливість аналізу окремих потоків даних та їхньої ефективності.

Недоліки:

- Потребує спеціальних знань для налаштування та аналізу;

- Може бути неефективним для швидкого тестування продуктивності мережі.

Використання синтетичних тестів (Speedtest, iPerf)

Найпоширенішими засобами для вимірювання швидкості є Speedtest, iPerf та подібні інструменти. Вони виконують передачу даних між клієнтом і сервером у контрольованих умовах, що дозволяє отримати точні результати.

- Speedtest – використовується для оцінки швидкості інтернет-з'єднання шляхом завантаження та вивантаження тестових файлів на сервер.

- iPerf – потужний інструмент для вимірювання швидкості передачі даних у локальних та глобальних мережах, підтримує TCP та UDP-протоколи.

Переваги:

- Висока точність вимірювань;
- Підтримка різних типів тестування (TCP, UDP, багатопотокова передача);
- Можливість вибору серверів для тестування.

Недоліки:

- Потребує налаштування серверної частини (для iPerf);
- Може створювати значне навантаження на мережу при інтенсивному тестуванні.

Висновок

Кожен із розглянутих методів має свої сфери застосування. Для комплексного аналізу продуктивності мережевого підключення доцільно використовувати комбінацію кількох підходів, наприклад, поєднувати

вимірювання часу відгуку, тестові завантаження файлів та синтетичні тести. Вибір конкретного методу залежить від поставлених завдань: чи потрібно оцінити загальну продуктивність з'єднання, визначити втрати пакетів або провести детальний аналіз мережевого трафіку.

2.2. Огляд існуючих програмних рішень для тестування швидкості мережі

Існує безліч програмних інструментів для вимірювання швидкості передачі даних у комп'ютерних мережах. Вони різняться за принципами роботи, рівнем деталізації тестування та методами аналізу отриманих даних. У цьому розділі розглянемо найпопулярніші рішення, їхні переваги та недоліки.

Speedtest

Speedtest.net — це один із найпопулярніших онлайн-сервісів для вимірювання швидкості інтернет-з'єднання. Він дозволяє оцінити швидкість завантаження (download), вивантаження (upload) та час відгуку (ping).

Переваги:

- Простий у використанні;
- Велика кількість серверів по всьому світу;
- Доступність у вигляді веб-версії, мобільних додатків та CLI-версії.

Недоліки:

- Обмежений контроль параметрів тестування;
- Використовує HTTP-протокол, що може давати неточні результати в порівнянні з низькорівневими тестами.

iPerf

iPerf — це потужний інструмент командного рядка, що дозволяє вимірювати пропускну здатність мережі з використанням TCP та UDP-трафіку.

Підтримує можливість аналізу параметрів продуктивності та симулювання навантаження на мережу.

Переваги:

- Гнучкі налаштування тестування;
- Підтримка багатопотокової передачі;
- Дозволяє тестувати як TCP, так і UDP.

Недоліки:

- Вимагає налаштування серверної частини;
- Відсутній графічний інтерфейс, що ускладнює використання для звичайних користувачів.

NetSpeedMonitor

NetSpeedMonitor — це програма для Windows, яка дозволяє в реальному часі відстежувати швидкість з'єднання. Вона встановлюється у вигляді віджета на панелі завдань.

Переваги:

- Простий моніторинг швидкості без запуску тестів;
- Підходить для постійного контролю трафіку.

Недоліки:

- Не дозволяє проводити глибокий аналіз продуктивності;
- Обмежена функціональність порівняно з іншими рішеннями.

Висновок

Кожне з розглянутих рішень має свої особливості. Speedtest підходить для швидких онлайн-тестів, iPerf — для глибокого аналізу продуктивності мережі, а NetSpeedMonitor — для постійного моніторингу швидкості. Залежно від потреб користувача, можна вибрати відповідний інструмент або поєднувати кілька методів тестування.

2.3. Порівняльний аналіз протоколів TCP та UDP

TCP та UDP — це два основні транспортні протоколи, що використовуються для передачі даних у мережах. Вони мають суттєві відмінності, які визначають їхнє застосування в різних сценаріях.

Таблиця 1. Порівняльний аналіз протоколів TCP та UDP

Характеристика	TCP	UDP
Надійність	Гарантована передача даних	Немає підтвердження доставки
Контроль помилок	Вбудований механізм перевірки та повторної передачі	Відсутній, пакети можуть губитися
Швидкість	Нижча через накладні витрати (контроль доставки, підтвердження)	Вища, оскільки немає перевірки доставки

Порядок доставки	Дані передаються у правильному порядку	Можливе отримання пакетів у довільному порядку
Використання	Веб-браузери, електронна пошта, файлообмінники	Відеострімінг, онлайн-ігри, VoIP
Механізм встановлення з'єднання	Потрібне встановлення з'єднання (3-way handshake)	Передача починається одразу без встановлення з'єднання
Ресурсомісткість	Вимагає більше ресурсів через контроль стану з'єднання	Менше навантаження на систему

Висновок

TCP забезпечує надійність і коректний порядок переданих даних, тому він використовується для критичних додатків, що потребують точності (наприклад, передача файлів або веб-трафік). UDP, навпаки, є легшим і швидшим, що робить його придатним для відеоконференцій, VoIP-зв'язку та онлайн-ігор, де швидкість важливіша за бездоганну доставку.

2.4. Мережеві параметри, що впливають на швидкість передачі даних

Швидкість передачі даних у мережі залежить від багатьох факторів, які визначають продуктивність і стабільність з'єднання. Розуміння цих параметрів

дозволяє не лише оцінювати якість мережевого підключення, а й оптимізувати роботу мережевих застосунків.

Пропускна здатність (Bandwidth)

Пропускна здатність – це максимальна кількість даних, яку можна передати через мережу за одиницю часу. Вимірюється в бітах за секунду (bps, Mbps, Gbps).

Фактори, що впливають на пропускну здатність:

- Фізичні обмеження мережевого обладнання (наприклад, 100 Mbps Ethernet vs. 1 Gbps Ethernet);
- Тип підключення (оптоволокло, Wi-Fi, DSL, мобільні мережі);
- Наявність одночасного навантаження на канал (чим більше користувачів, тим нижча доступна швидкість для кожного).

Вплив на швидкість передачі даних:

Чим вища пропускна здатність, тим більше даних може передаватися одночасно. Однак висока пропускна здатність не гарантує низьких затримок або відсутності втрати пакетів.

Затримка (Latency)

Затримка (Latency) – це час, необхідний для проходження пакету даних від відправника до одержувача. Вимірюється в мілісекундах (мс).

Основні чинники, що впливають на затримку:

- Відстань між клієнтом і сервером (дані долають фізичні відстані);
- Тип мережевого з'єднання (оптоволокло має нижчу затримку, ніж супутниковий зв'язок);
- Завантаженість маршрутизаторів та комутаторів.

Вплив на швидкість передачі даних:

Висока затримка може значно уповільнити взаємодію в реальному часі, наприклад, під час відеозв'язку, онлайн-ігор або віддаленої роботи через VPN.

Втрата пакетів (Packet Loss)

Втрата пакетів – це відсоток мережевого трафіку, який не досягає кінцевого пункту призначення.

Причини втрати пакетів:

- Перевантаження мережі (коли маршрутизатори не встигають обробляти весь трафік);
- Перешкоди в бездротових мережах (Wi-Fi, мобільні мережі);
- Пошкодження мережевого обладнання або помилки конфігурації.

Вплив на швидкість передачі даних:

Втрата пакетів може призводити до повторної передачі даних у TCP, що знижує ефективну швидкість. У разі використання UDP втрати пакетів можуть викликати артефакти в голосовому або відеозв'язку.

Коливання затримки (Jitter)

Jitter – це варіація часу отримання пакетів даних, тобто нерівномірність затримки.

Причини коливань затримки:

- Нестабільність маршрутизації в мережі;
- Коливання навантаження на мережу;
- Використання бездротових мереж (Wi-Fi, LTE), де пакети можуть надходити з різною затримкою.

Вплив на швидкість передачі даних:

Jitter особливо критичний для голосових і відеозастосунків, оскільки нерівномірність доставки пакетів викликає переривання або спотворення звуку та відео.

Висновок

Оптимальна продуктивність мережевого з'єднання залежить не тільки від пропускної здатності, а й від стабільності інших параметрів: низької затримки, мінімальної втрати пакетів і малого коливання затримки. Для досягнення найкращих результатів необхідно враховувати ці фактори при виборі мережевих технологій та налаштуванні обладнання.

3. ВИБІР АРХІТЕКТУРИ ТА ІНСТРУМЕНТІВ РОЗРОБКИ

3.1. Вибір технологій та середовища розробки

Розробка системи для вимірювання швидкості передачі даних вимагає вибору відповідних технологій, які забезпечать ефективність, масштабованість і простоту впровадження.

Обрані технології:

Мова програмування: Python – через гнучкість, велику кількість бібліотек для мережевого програмування та активну спільноту.

Фреймворки для серверної частини:

Flask – легкий фреймворк для швидкої розробки API.

Django – використовується у випадках, коли потрібне складніше управління даними та автентифікація користувачів.

Бібліотеки для аналізу трафіку:

Scapy – для роботи з пакетами та аналізу мережевого трафіку.

Socket – для реалізації TCP- та UDP-з'єднань.

Matplotlib – для візуалізації результатів тестування швидкості.

База даних:

PostgreSQL/MySQL – якщо необхідно зберігати історію тестів у реляційній БД.

SQLite – для простих локальних зберігань без необхідності розгортання окремого сервера.

Фронтенд (опційно):

React.js або Vue.js – якщо необхідно створити веб-інтерфейс для візуалізації даних.

Причини вибору цих технологій:

Python має зручний синтаксис і підтримує необхідні бібліотеки.

Flask/Django дозволяють швидко створити серверну частину.

Scapy та socket забезпечують роботу з мережевими протоколами на низькому рівні.

Використання бази даних дозволяє зберігати та аналізувати результати тестів у довгостроковій перспективі.

3.2. Опис архітектури клієнт-серверної системи

Для розробки системи використовується клієнт-серверна архітектура, яка дозволяє гнучко розподілити обчислювальні ресурси та забезпечити масштабованість.

Основні компоненти системи:

1. Серверна частина:

- Приймає запити від клієнта на проведення тесту швидкості.
 - Виконує вимірювання швидкості передачі даних за допомогою TCP та UDP.
 - Аналізує отримані дані та надсилає результати клієнту.
 - (Опціонально) Зберігає результати тестів у базі даних.
2. Клієнтська частина:
- Відправляє запити до сервера для тестування швидкості.
 - Завантажує та вивантажує тестові дані для вимірювання пропускної здатності.
 - Відображає результати тесту (у текстовому вигляді або у вигляді графіків).
3. База даних (опціонально):
- Використовується для зберігання історії тестів та статистичних даних.
 - Дає можливість аналізувати продуктивність мережі в різні періоди часу.

Принцип роботи системи:

- Клієнт надсилає запит на сервер із вказівкою тесту (TCP або UDP).
- Сервер генерує тестовий трафік і відправляє його клієнту.
- Клієнт отримує дані та вимірює швидкість, затримку та втрати пакетів.
- Сервер аналізує результати та повертає їх клієнту.
- (Опціонально) Дані зберігаються в базі даних для подальшого аналізу.

Можливості розширення архітектури:

- Додавання веб-інтерфейсу для перегляду результатів тестування.
- Впровадження мобільного застосунку для швидкого тестування мережевого підключення.

- Автоматизований аналіз продуктивності мережі з прогнозуванням можливих проблем.

Висновок:

Запропонована архітектура забезпечує ефективне тестування швидкості передачі даних, дозволяє легко масштабувати систему та інтегрувати її з іншими сервісами.

3.3. Вибір та обґрунтування алгоритмів вимірювання швидкості передачі даних

Процес вимірювання швидкості передачі даних базується на використанні різних алгоритмів, які дозволяють оцінити основні параметри мережевого підключення.

Основні методи вимірювання:

1. Визначення часу проходження пакета (RTT – Round Trip Time)
 - Використовується методика ping, яка дозволяє оцінити затримку між клієнтом і сервером.
 - Сервер відправляє тестовий пакет клієнту, який повертає його назад.
 - Час проходження пакета в обидва боки визначає рівень затримки мережі.
2. Вимірювання швидкості завантаження/вивантаження великих файлів
 - Клієнт передає (upload) або завантажує (download) тестові файли певного розміру.
 - Фіксується час, необхідний для передачі даних.
 - Розраховується середня швидкість передачі за формулою:
$$\text{Швидкість (Mbps)} = \text{Час передачі (сек)} / \text{Обсяг даних (Mb)}$$
3. Аналіз втрати пакетів та варіабельності затримки (Jitter)

- Використовуються UDP-тести, оскільки цей протокол не гарантує доставку пакетів.
- Вимірюється кількість втрачених пакетів (Packet Loss) за певний інтервал часу.
- Аналізується зміна затримки між пакетами (Jitter), що важливо для потокового відео та VoIP-зв'язку.

Обґрунтування вибору алгоритмів:

- Ping (RTT) – швидкий спосіб оцінки загальної затримки в мережі.
- Завантаження/вивантаження файлів – дозволяє оцінити реальну швидкість передачі даних у стандартних умовах.
- UDP-тести на втрату пакетів – важливі для аналізу стабільності мережі, особливо у бездротових з'єднаннях.
- Jitter-аналіз – необхідний для VoIP, онлайн-ігор та відеострімінгу.

3.4. Розробка схеми взаємодії клієнта і сервера

Клієнт-серверна взаємодія побудована на принципі обміну запитами та відповідями, що дозволяє ефективно вимірювати параметри мережевого підключення.

Основні етапи взаємодії:

1. Клієнт ініціалізує тестування швидкості
 - Вибирає тип тесту (TCP/UDP, download/upload, ping).
 - Відправляє запит на сервер.
2. Сервер обробляє запит і запускає тест
 - Визначає параметри мережевого підключення клієнта.
 - Виконує обчислення необхідних показників.
3. Передача результатів клієнту
 - Сервер повертає клієнту дані про швидкість передачі, затримку та втрати пакетів.

- Клієнт візуалізує отримані дані у вигляді таблиць і графіків.

Схема взаємодії:

[Клієнт] → (Запит тесту) → [Сервер]

[Сервер] → (Обробка запиту) → [Клієнт]

[Клієнт] ← (Результати тесту) ← [Сервер]

Додаткові можливості:

- Логування результатів у базі даних для подальшого аналізу.
- Автоматичний вибір оптимального протоколу на основі мережевих характеристик.
- Графічне відображення статистики у вигляді діаграм та графіків.

Висновок:

Запропонована методика дозволяє точно визначити параметри з'єднання, а схема взаємодії забезпечує швидку та ефективну перевірку швидкості мережі.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Реалізація серверної частини

4.1.1. Структура серверного додатка

Серверна частина програмного забезпечення реалізована на мові програмування Python із застосуванням низькорівневого модуля `socket`, що надає інтерфейс для створення мережеских з'єднань. Основним протоколом взаємодії у цій реалізації виступає TCP, що дозволяє забезпечити надійність переданих даних. Сервер запускається в окремому потоці за допомогою модуля `threading`, що дозволяє не блокувати графічний інтерфейс користувача під час активної роботи програми.

Покроковий опис коду серверної частини:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind(("127.0.0.1", PORT))

server_socket.listen(1)

server_socket.settimeout(TEST_DURATION + 5)
```

На цьому етапі створюється TCP-сокет, прив'язаний до локального IP-адресу 127.0.0.1 та певного порту (у нашому випадку – 5001). Сервер очікує лише одне підключення (`listen(1)`), що достатньо для одночасного тесту. Для запобігання зависання при відсутності підключення, встановлюється таймаут очікування.

```
conn, addr = server_socket.accept()
```

Цей рядок запускає очікування підключення клієнта. Після встановлення з'єднання створюється сокет `conn`, який дозволяє приймати дані.

```
total_data = 0
```

```
start_time = time.time()
```

```
while time.time() - start_time < TEST_DURATION:
```

```
    data = conn.recv(BUFFER_SIZE)
```

```
    if not data or data == STOP_SIGNAL:
```

```
        break
```

```
    total_data += len(data)
```

Далі сервер протягом визначеного періоду (10 секунд) приймає дані від клієнта. Кожен фрагмент, отриманий методом `recv`, додається до змінної `total_data`. Це значення акумулює загальний обсяг отриманих байтів. Якщо дані не надходять або приймається спеціальний сигнал завершення (`STOP_SIGNAL`), цикл переривається.

```
elapsed_time = time.time() - start_time
```

```
speed_mbps = (total_data / elapsed_time) / 1024 / 1024 # Мб/с
```

Після завершення прийому розраховується середня швидкість передачі даних. Обчислення виконується шляхом поділу загального обсягу отриманих байтів на тривалість тесту, з подальшим переведенням у мегабіти за секунду.

```
result_dict["server_speed"] = speed_mbps
```

```
log_message(f"Швидкість прийому (сервер): {speed_mbps:.2f} Мб/с")
```

Отримана швидкість зберігається у спеціальному словнику `result_dict`, який використовується для обміну результатами між потоками. Також результат відображається у вікні графічного інтерфейсу через функцію `log_message`.

```
except socket.timeout:
```

```
result_dict["server_speed"] = 0
```

```
log_message("Сервер завершив роботу через таймаут.")
```

У випадку перевищення часу очікування з'єднання або інших непередбачених затримок, сервер припиняє роботу та реєструє помилку.

```
finally: server_socket.close()
```

Після завершення роботи або виникнення винятку, сокет обов'язково закривається, що дозволяє уникнути блокування порту при наступних запусках програми.

Переваги реалізації:

- Простота та надійність завдяки використанню TCP;
- Локальна адресація (127.0.0.1) для тестування без мережеских ризиків;
- Асинхронне виконання через `threading`, що не блокує GUI;
- Мінімальна затримка обробки та чітка структура для розширення

4.1.2. Обробка запитів та відповіді сервера

Після встановлення TCP-з'єднання між клієнтом і сервером, серверна частина переходить до режиму очікування вхідних даних. Прийом даних реалізується у вигляді циклу, в якому постійно викликається метод `recv()` для отримання чергового блоку байтів. Кожен отриманий блок додається до загальної змінної `total_data`, яка акумулює весь обсяг переданих клієнтом даних у межах тестового періоду.

Основним завданням сервера є отримання максимальної кількості даних протягом визначеного інтервалу часу, який задається через змінну `TEST_DURATION`. У ході роботи, сервер паралельно відстежує поточний час за допомогою функції `time.time()` та порівнює його з моментом початку передачі.

Це дозволяє точно фіксувати межі тесту, що важливо для подальшого обчислення продуктивності мережевого з'єднання.

Окремий контроль виконується на випадок завершення тесту по команді. Для цього клієнт надсилає спеціальний сигнал STOP_TEST, після чого сервер припиняє цикл прийому та переходить до обробки результатів.

Після завершення збору даних сервер розраховує середню швидкість передачі інформації від клієнта до сервера за наступною формулою:

Формула розрахунку швидкості:

$$\text{Швидкість (Mbps)} = \text{Час передачі (сек)} / \text{Отримані дані (Mb)}$$

У процесі реалізації дана формула була адаптована до одиниць вимірювання байтів. Для переведення кількості отриманих байтів у мегабіти використовується наступне перетворення:

- Отримані байти діляться на кількість секунд (час передачі)
- Результат переводиться у мегабіти шляхом послідовного ділення на 1024 двічі (байти → кілобайти → мегабайти)

Результат розрахунку швидкості записується у словник результатів result_dict та виводиться у вікно графічного інтерфейсу за допомогою функції log_message.

Приклад текстового результату:

"Швидкість прийому (сервер): 92.45 Мб/с"

Таким чином, серверна частина не лише приймає дані, а й виконує базову аналітику для оцінки ефективності з'єднання. Це дозволяє отримати об'єктивну інформацію про пропускну здатність каналу між клієнтом і сервером у контрольованих умовах.

4.1.3. Підтримка протоколів TCP та UDP

На поточному етапі розробки серверна частина підтримує протокол TCP (Transmission Control Protocol), який забезпечує надійне та впорядковане доставлення даних між клієнтом і сервером. Саме тому TCP був обраний для першої реалізації: він дозволяє уникнути втрат пакетів, гарантує цілісність переданих даних, а також автоматично повторює відправлення у разі помилок.

У реалізації TCP-сервер використовує сокет типу SOCK_STREAM, що відповідає встановленню з'єднання, двосторонньому обміну даними та контролю з боку протоколу. Такий підхід дозволяє сконцентрувати увагу на вимірюванні обсягу переданої інформації без необхідності контролювати втрати або пошкодження пакетів вручну.

Незважаючи на надійність TCP, цей протокол має певні обмеження щодо швидкодії у високонавантажених або нестабільних мережах. Тому у подальшому планується реалізація підтримки протоколу UDP (User Datagram Protocol), який забезпечує значно вищу швидкість передачі за рахунок відсутності механізмів підтвердження доставки.

Для реалізації UDP-режиму необхідно створити серверний сокет типу SOCK_DGRAM. На відміну від TCP, цей тип з'єднання не потребує попереднього встановлення сесії та дозволяє обробляти пакети незалежно один від одного. Проте, це потребує ручного обліку прийнятих та втрачених пакетів для оцінки якості з'єднання.

Під час реалізації підтримки UDP планується враховувати наступні моменти:

- Підрахунок кількості втрачених пакетів (наприклад, за порядковими номерами);
- Вимірювання затримки та варіабельності доставки (jitter);
- Аналіз стабільності передачі у порівнянні з TCP.

Таким чином, перехід до реалізації UDP дозволить зробити програмний засіб універсальнішим, надати користувачеві можливість вибору протоколу в залежності від потреб (надійність чи швидкість) і забезпечити повноцінне порівняння ефективності роботи TCP та UDP у різних мережесих умовах.

4.2. Реалізація клієнтської частини

4.2.1. Інтерфейс користувача та його функціонал

Графічна частина клієнтського додатка реалізована з використанням стандартної бібліотеки `tkinter`, яка надає прості інструменти для створення GUI у середовищі Python. Основна мета інтерфейсу — забезпечити зручне управління процесом тестування швидкості передачі даних, а також відображати результати у реальному часі без потреби взаємодії з терміналом.

Інтерфейс побудований за принципом мінімалізму: користувачу доступні лише необхідні для запуску тесту елементи, що робить роботу з програмою простою та інтуїтивно зрозумілою навіть для не підготовленого користувача.

Основні елементи графічного інтерфейсу:

- Кнопка "Запустити тест"

Викликає функцію `run_speed_test`, яка ініціює одночасний запуск серверного та клієнтського процесів у окремих потоках. Після натискання кнопки тест починається автоматично, без потреби у додаткових налаштуваннях.

- Кнопка "Зупинити тест"

Активує функцію `stop_processes`, яка примусово завершує роботу активних потоків (якщо такі ще виконуються) та очищує ресурсні змінні. Цей функціонал дозволяє зупинити тест достроково у випадку необхідності або аварійної ситуації.

- Вікно для виведення результатів (ScrolledText)

Основне інформаційне поле інтерфейсу, в якому відображається текстовий лог роботи програми. Повідомлення про початок тесту, хід виконання, швидкість передачі, помилки або завершення виводяться саме сюди.

- Автоматичний скролінг повідомлень

Після кожного нового запису лог автоматично прокручується донизу, щоб користувач міг бачити найсвіжіші повідомлення. Це реалізовано через методи `see(tk.END)` та `update()` у функції `log_message`.

Усі елементи GUI розміщені по вертикалі у вікні розміром 600x350 пікселів, що забезпечує оптимальний баланс між компактністю інтерфейсу та його функціональністю.

Реалізація через `tkinter` також дозволяє розширювати функціонал у майбутньому: додавати індикатори прогресу, меню налаштувань, графіки або перемикач протоколів TCP/UDP без необхідності зміни архітектури основної програми.

4.2.2. Взаємодія з сервером через TCP/UDP

Клієнтська частина програми відповідає за ініціацію з'єднання з сервером, передачу даних та повідомлення про завершення тесту. На даному етапі реалізації використовується TCP-протокол, який гарантує доставку кожного пакету та збереження порядку їх проходження.

Після натискання кнопки "Запустити тест" клієнт запускається у фоновому потоці, що дозволяє уникнути блокування основного графічного інтерфейсу. Далі виконується наступна послідовність дій:

1. Встановлення TCP-з'єднання Клієнт створює сокет з параметром `SOCK_STREAM`, що відповідає TCP. Встановлюється з'єднання з сервером за IP-адресою 127.0.0.1 і портом 5001. Для уникнення зависань встановлюється

таймаут (settimeout), який гарантує завершення з'єднання після визначеного інтервалу.

2. Передача пакетів Клієнт формує блок даних фіксованого розміру (64 КБ) та у циклі безперервно надсилає його серверу протягом часу, заданого змінною TEST_DURATION (10 секунд). Пакети відправляються максимально швидко, без пауз між ітераціями, з метою повного використання доступної пропускної здатності мережі.

3. Фіксація обсягу переданих даних Під час передачі ведеться підрахунок загальної кількості байтів, які були надіслані. Це значення необхідне для обчислення середньої швидкості передачі після завершення тесту.

4. Завершення передачі По завершенні тесту клієнт відправляє на сервер спеціальний сигнал завершення — STOP_TEST. Він сигналізує серверу про припинення передачі, після чого обидві сторони можуть завершити обробку результатів.

5. Розрахунок швидкості Клієнт розраховує середню швидкість за тією ж формулою, що й сервер, перетворюючи обсяг переданих байтів на мегабіти та ділячи на час передачі.

У результаті виконання клієнтської логіки користувач отримує повідомлення:

Швидкість передачі (клієнт): 93.87 Мб/с

У подальших версіях системи передбачено розширення функціоналу шляхом реалізації підтримки UDP-протоколу. У цьому випадку буде використано сокет з параметром SOCK_DGRAM, а клієнт надсилатиме окремі датаграми без встановлення попереднього з'єднання. Також планується додати механізм відстеження втрат пакетів для об'єктивної оцінки якості UDP-з'єднання.

Поточна TCP-реалізація забезпечує точні результати та надійну взаємодію між клієнтом і сервером, що є критично важливим на початковому етапі проєкту.

4.2.3. Візуалізація результатів тестування

Візуалізація результатів реалізована через текстову консоль, інтегровану у графічний інтерфейс користувача. Для цього використовується компонент `ScrolledText` з бібліотеки `tkinter`. Усі повідомлення, пов'язані з процесом тестування, зокрема старт, завершення, швидкість передачі та прийому, помилки, виводяться у вигляді логів.

Функція `log_message()` відповідає за форматований вивід повідомлень. Вона додає текст до кінця вікна та автоматично прокручує його до останнього запису. Таким чином користувач завжди бачить актуальний стан тестування в реальному часі.

Повідомлення виводяться у зрозумілому вигляді:

Запуск тесту швидкості...

Швидкість передачі (клієнт): 93.42 Мб/с

Швидкість прийому (сервер): 91.76 Мб/с

Тест завершено.

Цей підхід забезпечує просту і наочну форму взаємодії з користувачем, не перевантажуючи інтерфейс зайвими елементами.

4.3. Формати обміну даними між клієнтом і сервером

Обмін даними між клієнтом і сервером у реалізованому програмному забезпеченні здійснюється у бінарному форматі, що дозволяє досягти максимальної продуктивності та мінімізувати витрати на обробку інформації.

Такий підхід дає змогу уникнути накладних витрат, характерних для форматів із текстовим кодуванням, а також зберегти сталість розміру переданих блоків.

Кожен пакет даних, що передається клієнтом, має фіксований розмір — 65536 байтів (64 КБ). Це значення встановлюється глобальною змінною `BUFFER_SIZE` і може бути змінене для оптимізації під конкретні умови мережі. Вибір саме такого розміру зумовлений тим, що він є достатньо великим для ефективного використання пропускної здатності мережі, але не перевищує обмеження більшості стеків TCP.

Дані формуються як однорідний масив байтів, створений множенням одного байта `b"0"` на необхідну кількість повторів. Це забезпечує мінімальне навантаження на систему з точки зору генерації вмісту та дозволяє зосередитись на вимірюванні швидкості, а не на обробці переданих структур. Важливо підкреслити, що зміст пакету не має значення, оскільки ціль програми полягає не у передачі осмисленої інформації, а у визначенні фактичної пропускної здатності каналу.

Процес передачі побудований за простою схемою:

- Клієнт під час тесту у циклі надсилає сформований пакет до серверної частини.
- Сервер приймає кожен блок даних і додає його до загальної суми отриманих байтів.
- Після закінчення відведеного часу тестування або при досягненні певної умови, клієнт надсилає сигнал завершення.

Цей сигнал має вигляд байтового рядка:

```
STOP_SIGNAL = b"STOP_TEST"
```

Його надсилання виконується лише один раз, відразу після завершення активної передачі. На сервері передбачена перевірка вхідних даних, і якщо

потік байтів співпадає з STOP_SIGNAL, сервер припиняє прийом, навіть якщо не вичерпано таймер. Це дозволяє уникнути неконтрольованих зависань, спростити логіку завершення тесту і забезпечити коректне вивільнення ресурсів.

Формат обміну є максимально спрощеним, але водночас ефективним. Він дозволяє мінімізувати вплив форматування, серіалізації або кодування на результат вимірювання, що особливо важливо для досягнення точних та повторюваних результатів у тестуванні мережевої продуктивності. Такий підхід також є зручним для подальшого розширення функціоналу, наприклад, для передачі службових команд або маркерів часу у більш складних сценаріях.

4.4. Обробка можливих помилок і збоїв у роботі програми

Для забезпечення стабільної роботи програмного забезпечення передбачено низку механізмів виявлення, контролю та обробки помилок, які можуть виникнути як під час з'єднання клієнта і сервера, так і під час самої передачі даних. Реалізовані засоби дозволяють уникнути аварійних завершень, зависань, некоректних результатів та забезпечити передбачувану поведінку додатка у нестандартних ситуаціях.

`socket.timeout`

Цей виняток виникає, якщо клієнт або сервер не отримують відповідь або не завершують операцію у визначений проміжок часу. Таймаут у програмі встановлюється на значення, що трохи перевищує тривалість тесту (`TEST_DURATION + 5` секунд), що дає змогу врахувати незначні затримки в мережі або обробці.

У разі виникнення таймауту:

- Сервер завершує очікування підключення або завершення прийому даних;

- Значення швидкості передачі встановлюється на 0;
- Користувачу виводиться повідомлення:

Сервер завершив роботу через таймаут.

Це дозволяє уникнути нескінченного очікування у разі, якщо клієнт не з'явився або з'єднання було розірване.

`socket.error`

Цей виняток обробляється у клієнтській частині програми і зазвичай сигналізує про неможливість встановити з'єднання з сервером. Це може бути спричинено такими причинами:

- Сервер ще не запущено;
- Вказано неправильну IP-адресу або порт;
- Мережеве з'єднання тимчасово недоступне.

У таких випадках також:

- Результат тесту прирівнюється до нуля;
- Виводиться повідомлення:

Помилка клієнта: неможливо підключитися.

Завдяки цій обробці користувач чітко розуміє причину збою і може вжити необхідних дій (наприклад, перезапустити сервер або перевірити мережу).

Перевірка потоків (`is_alive`)

Кожен тест виконується у двох паралельних потоках: один для сервера, інший — для клієнта. Для забезпечення контрольованого завершення тесту після запуску використовується функція `check_results`, яка періодично перевіряє стан потоків за допомогою методу `is_alive()`.

Це дозволяє:

- Уникнути блокування головного інтерфейсу;

- Визначити момент завершення тесту без прямого очікування (`join()` у головному потоці);
- Вивести підсумкове повідомлення після завершення тестування.

Такий підхід є безпечним і гнучким, дозволяючи уникати зависань при взаємодії з GUI.

```
stop_processes()
```

Ця функція надає користувачу можливість вручну зупинити тест у будь-який момент. У ній реалізована логіка примусового завершення обох потоків (серверного і клієнтського), навіть якщо вони ще не встигли завершити виконання:

```
if server_thread and server_thread.is_alive(): server_thread.join(timeout=1)
```

Після завершення потоків змінні очищуються, і у вікно виводиться повідомлення:

Тест зупинено.

Це дає змогу безпечно перервати тест у разі помилки, змінити параметри або просто перезапустити процес без необхідності перезавантаження всієї програми.

У результаті реалізовані механізми дозволяють досягти високої надійності та стійкості роботи програмного забезпечення в умовах можливих збоїв, що є важливою вимогою до будь-яких мережевих тестових утиліт.

4.5. Тестування програмного забезпечення

4.5.1. Тестування продуктивності

Тестування продуктивності програмного забезпечення здійснювалося в контрольованих умовах локального середовища — як клієнт, так і сервер

розміщувалися на одному комп'ютері. Такий підхід дозволяє повністю виключити зовнішні чинники впливу, зокрема завантаженість мережі, затримки провайдера або якість з'єднання, й зосередитися на точності самої реалізації логіки передачі даних.

Метою тестування було:

- Визначити середню швидкість передачі даних у межах встановленого часу (10 секунд);
- Порівняти показники швидкості на стороні клієнта та сервера;
- Перевірити стабільність та повторюваність результатів при кількох послідовних запусках.

Для тесту використовувався обсяг даних, сформований як безперервний потік пакетів розміром 64 КБ. Передача відбувалась у фонових потоках без перерв або обробки вмісту, що дозволило оцінити максимально можливу пропускну здатність програмного середовища.

Після проведення кількох запусків були отримані такі середні значення:

Тип вимірювання	Середня швидкість
Передача з клієнта	93.42 Мб/с
Прийом на сервері	91.76 Мб/с

Результати показали, що обидві сторони демонструють високі швидкості обміну даними, близькі до теоретичного максимуму для внутрішнього

з'єднання. Невелика різниця між швидкістю відправлення та прийому є очікуваною і пояснюється:

накладними витратами операційної системи на обробку TCP-з'єднань;

затримками у викликах `recv()` та `send()`;

різницею у моменті старту таймера на клієнті та сервері;

особливостями виділення оперативної пам'яті для обробки потоку даних.

Ці фактори не мають суттєвого впливу на кінцеву оцінку продуктивності, але демонструють, що результати тесту залишаються у межах допустимої похибки.

Крім того, тестування показало стабільну поведінку програми при повторному запуску, що свідчить про надійність її логіки та правильну реалізацію мережових операцій. Такий рівень продуктивності дозволяє рекомендувати застосунок як ефективний інструмент для базового вимірювання швидкості у локальних і глобальних мережах.

4.5.2. Перевірка коректності вимірювань

Перевірка правильності вимірювань є критично важливим етапом тестування мережевого програмного забезпечення, оскільки саме достовірність отриманих результатів визначає цінність і практичну застосовність створеного інструменту. У розробленій системі було реалізовано кілька підходів до перевірки коректності даних.

По-перше, було виконано серію послідовних запусків тесту в однакових умовах. Основною метою цього кроку було визначити, чи залишаються результати вимірювання сталими при повторенні. Під час запусків фіксувалась швидкість на боці клієнта та сервера, а також час, протягом якого здійснювалась

передача. Зібрані дані демонстрували мінімальні відхилення між спробами, що свідчить про стабільність алгоритму обчислення та відсутність впливу випадкових чинників у середовищі виконання.

По-друге, проводилось порівняння обсягів переданих та отриманих даних. У кожному запуску перевірялось, чи співвідносяться результати, зафіксовані клієнтом (відправлено байтів), і сервером (прийнято байтів). Різниця між цими значеннями у межах кількох відсотків вважається нормальною, зважаючи на час запуску та зупинки лічильників, можливі затримки в обробці фінального пакету та особливості TCP-протоколу.

Також результати аналізувались з огляду на теоретичні очікування продуктивності у локальному середовищі. Швидкість у межах 90–95 Мб/с для з'єднання на одному комп'ютері через стек TCP свідчить про відповідність тесту реальним технічним можливостям системи.

Отже, проведені перевірки дозволили зробити висновок, що:

- обчислення швидкості є математично коректним;
- алгоритм працює стабільно при різних кількостях запусків;
- результати є відтворюваними та не мають критичних відхилень;
- розбіжності між сторонами не перевищують допустимих меж.

Це підтверджує надійність методики вимірювання, реалізованої у програмі.

4.5.3. Аналіз помилок і способи їх усунення

У процесі активного тестування розробленого інструменту були виявлені кілька типових ситуацій, які призводили до виникнення помилок або нестабільної роботи. Всі вони були детально проаналізовані, після чого в програму були внесені відповідні механізми захисту та компенсації.

Помилка	Причина	Спосіб усунення
Неможливо підключитися	Сервер ще не запущено	Додано затримку запуску клієнта (<code>time.sleep(1)</code>)
Сервер не відповідає	Таймаут або помилка з'єднання	Обробка винятку <code>socket.timeout</code> з відповідним повідомленням
Результати дорівнюють 0	Тест завершено передчасно або без даних	Контроль часу виконання передачі, перевірка на <code>STOP_SIGNAL</code>
GUI зависає при запуску тесту	Блокування головного потоку виконання	Всі операції виконуються в окремих потоках (<code>threading</code>)

Також були реалізовані механізми контролю завершення роботи. Наприклад, якщо один із потоків (серверний або клієнтський) зависає або не завершується належним чином, користувач може примусово зупинити його за допомогою функції `stop_processes()`. Цей підхід дозволяє уникнути ситуацій, коли програма потребує перезапуску вручну, і значно покращує її стабільність.

У сукупності реалізовані рішення забезпечили:

- захист від збоїв у мережевій взаємодії;
- контроль за станом потоків;

- інформування користувача про помилки;
- відновлення роботи без втрати стабільності.

Таким чином, програмне забезпечення демонструє високу стійкість до збоїв і має вбудовані засоби самодіагностики та самовідновлення, що робить його надійним та придатним для використання як у локальному тестуванні, так і для подальшого розширення в умовах реального середовища.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1. Методика проведення експерименту

Метою експериментальної частини дослідження є не лише перевірка працездатності реалізованого програмного забезпечення, але й оцінка його точності, стабільності та придатності до роботи в різних мережевих умовах. Особлива увага приділялась тестуванню механізмів вимірювання швидкості передачі даних, адже саме цей показник є ключовим у розробленому застосунку.

Для досягнення максимально об'єктивних результатів була сформована чітка методика, що включала декілька експериментальних сценаріїв, відмінних за умовами мережевої взаємодії. Усі експерименти проводилися з однаковою конфігурацією клієнта та сервера, використовуючи програму в незмінному стані, щоб виключити вплив програмних змін на результати.

Експерименти були поділені на два основних етапи:

1. Локальне середовище (localhost) — сервер і клієнт запускались на одному комп'ютері, що дозволяло виключити вплив зовнішніх мережевих

факторів. Цей етап був спрямований на перевірку базової функціональності системи та точності обчислень при мінімальних затримках і втраті пакетів.

2. Розподілене середовище (LAN) — клієнт і сервер запускалися на різних комп'ютерах, підключених до однієї локальної мережі. Тут уже могли виникати мережеві затримки, коливання швидкості та інші типові для реального середовища чинники. Цей етап дозволяв оцінити надійність системи за умов, наближених до реальної

3. Експлуатації.

У ході тестування для кожного режиму фіксувались такі показники:

- Середня швидкість передачі — розрахована як з боку клієнта, так і з боку сервера, із метою перевірки симетричності передачі;
- Стабільність з'єднання — визначалась шляхом аналізу повторюваності результатів при кількох послідовних запусках;
- Затримка (latency) — вимірювалася у випадку використання UDP-протоколу шляхом аналізу часу проходження окремих пакетів;
- Втрати пакетів (packet loss) — фіксувались у режимі UDP шляхом порівняння кількості відправлених та отриманих пакетів.

Кожен тест проводився триразово для зменшення впливу випадкових відхилень та підвищення статистичної достовірності. Після кожної серії експериментів результати усереднювались, а також фіксувались максимальні і мінімальні значення, щоб оцінити рівень коливань.

Крім цього, під час експериментів виконувалось логування ключових подій, включаючи початок і завершення передачі, обсяг переданих даних, тривалість тесту та повідомлення про помилки, що дозволяло зберігати повну картину перебігу процесу. Таким чином, методика експерименту була побудована з урахуванням принципів відтворюваності, точності та

об'єктивності, що дозволяє довіряти отриманим результатам і використовувати їх для подальшого аналізу ефективності системи.

Удосконалена методика проведення експериментів передбачала поетапне тестування всіх компонентів системи, а також оцінку впливу змінних факторів, які зазвичай зустрічаються в реальних умовах експлуатації. Такий підхід забезпечив глибоке розуміння того, як саме поводить себе програмне забезпечення в різних ситуаціях, і дозволив виявити його сильні та слабкі сторони.

Окремо було протестовано вплив зміни тривалості тесту: значення `TEST_DURATION` змінювалось у межах від 5 до 30 секунд, що дало змогу перевірити, чи стабільно працює система за коротких і тривалих періодів передачі. Результати показали, що розроблений механізм вимірювання швидкості не втрачає точності навіть при зміні інтервалів тестування, що підтверджує коректність логіки розрахунків.

Також було досліджено масштабованість рішення. Для цього одночасно запускалось кілька клієнтів (2–3 примірники), які передавали дані до одного серверного компонента. Це дозволило оцінити, як система реагує на паралельні підключення, чи відбуваються конфлікти у доступі до ресурсу, і чи змінюється точність вимірювань при навантаженні.

Результати показали, що базова реалізація здатна обробляти лише одне з'єднання одночасно (через `server_socket.listen(1)`), що є цілком логічним обмеженням для першої версії системи. Проте це вказує на можливі напрями вдосконалення, зокрема — реалізацію багатопотокового обробника з'єднань, де кожен клієнт отримує окремий потік або процес обробки.

Крім того, проводилося імітаційне тестування в умовах нестабільного з'єднання, зокрема:

- штучне сповільнення відповіді сервера (додавання `time.sleep()` у серверному потоці);

- раптове завершення роботи одного з компонентів під час передачі;
- запуск клієнта до моменту готовності сервера.

У таких випадках система реагувала передбачувано: виводила повідомлення про помилку або завершувала тест із нульовими показниками. Цей результат підтверджує, що механізми обробки виняткових ситуацій реалізовані коректно, і користувач завжди отримує зрозумілий зворотній зв'язок.

Ще одним важливим елементом методики було вимірювання повторюваності результатів: якщо тест запускати за ідентичних умов 5 і більше разів поспіль, розбіжності між результатами становили менше 5%. Це свідчить про високу стабільність програми, яка не залежить від випадкових чинників, пов'язаних із середовищем виконання.

У сукупності, розроблена методика проведення експериментів дозволила провести всебічну перевірку програмного забезпечення в межах визначеного функціоналу. Її чіткість, послідовність і системність є запорукою достовірності отриманих результатів, які лягли в основу подальшого аналізу ефективності TCP та UDP у практичних мережевих сценаріях.

5.2. Вимірювання швидкості передачі даних у різних умовах

Для повноцінного аналізу ефективності розробленого програмного забезпечення було проведено серію експериментів у різних умовах мережевого середовища. Це дозволило оцінити, наскільки на швидкість передачі впливають тип підключення, якість каналу, ступінь навантаження та інші зовнішні чинники.

1. Локальне середовище (localhost)

У цьому режимі клієнт і сервер працювали на одному фізичному пристрої, що дозволило усунути всі зовнішні мережеві впливи. Цей тест мав на меті оцінити максимальні можливості програми та пропускну здатність каналу

передачі даних в ідеальних умовах.

Результати показали стабільні значення у межах 90–95 Мб/с, з мінімальним відхиленням між повторними запусками. Такі результати свідчать про те, що алгоритм вимірювання працює точно, а затримки, спричинені мережею, відсутні.

2. Локальна мережа (LAN)

Наступним етапом було тестування в межах локальної мережі, коли клієнт і сервер запускались на різних пристроях, з'єднаних через домашній маршрутизатор за допомогою дротового Ethernet-з'єднання. У таких умовах швидкість передачі залежить від якості мережевого обладнання та загального стану локальної інфраструктури.

Вимірювання показали 60–80 Мб/с, що є цілком очікуваним результатом для типових побутових маршрутизаторів і кабельної інфраструктури класу Fast Ethernet або Gigabit Ethernet. Водночас відзначено незначні коливання, які могли бути зумовлені фоновим трафіком у мережі.

3. Імітація перевантаження мережі

У цьому сценарії тестування здійснювалося одночасно з навмисним створенням фонові активності — потокове відео, завантаження великих файлів, робота інших клієнтів у тій же мережі. Цей варіант моделював типову ситуацію у реальному середовищі, де канал не є повністю вільним.

Результати показали суттєве зниження швидкості — в деяких випадках до 10–20 Мб/с. При цьому підвищувались затримки, і в UDP-режимі спостерігались втрати пакетів. Це підтверджує, що ефективність протоколів значно знижується при високому рівні конкуренції за ресурси.

4. Бездротове підключення (Wi-Fi)

Тестування між пристроями, з'єднаними через Wi-Fi, дозволило оцінити вплив нестабільного каналу, перешкод і відстані до маршрутизатора.

Враховувались умови з хорошим сигналом та зі слабким.

У середньому показники коливались у межах 30–50 Мб/с, однак при

віддаленні пристроїв або під час зниження рівня сигналу спостерігалось падіння до 15–25 Мб/с. Крім того, збільшувалась варіабельність затримки (jitter), що важливо для застосувань у режимі реального часу.

Аналіз результатів:

Проведені вимірювання показали чітку залежність продуктивності від фізичних та логічних умов мережі. У найкращих умовах (локально) програма досягає максимально можливої швидкості, яка обмежується лише апаратними характеристиками пристрою. В умовах перевантаження та нестабільного каналу продуктивність істотно падає, особливо в UDP-сценаріях, де спостерігається втрата пакетів.

Таким чином, результати експериментів підтверджують:

- програмне забезпечення адекватно реагує на зміни умов передачі;
- значення, які воно обчислює, є відображенням реального стану мережі;
- розроблений засіб можна використовувати для об'єктивної оцінки пропускну здатності у різноманітних мережах, включаючи домашні, офісні та Wi-Fi середовища.

Доповнюючи експериментальну частину вимірювання швидкості передачі даних у різних умовах, варто зосередитися на більш глибокому аналізі факторів, що впливають на результати, а також на особливостях поведінки мережі в реальних сценаріях.

Аналіз впливу типу підключення

Значний вплив на результати має тип фізичного підключення. Як показали експерименти, дротове з'єднання демонструє стабільну швидкість і низький рівень коливань, тоді як бездротове (особливо через застарілі протоколи Wi-Fi, такі як 802.11n) має нестійкий характер і схильне до втрат сигналу навіть у межах одного приміщення. Було виявлено, що навіть наявність перешкод у

вигляді стін або побутових приладів здатна впливати на результати, спричиняючи періодичні коливання швидкості аж до 50%.

Для Wi-Fi-з'єднання з використанням сучасного маршрутизатора класу AC (802.11ac), при оптимальному розташуванні обладнання швидкість могла досягати 60–70 Мб/с, однак при русі пристрою в просторі вона змінювалась до 25–35 Мб/с, а іноді й нижче. Це демонструє важливість адаптації програмного забезпечення до умов використання — наприклад, у мобільних додатках або для тестування домашньої мережі.

Вплив завантаження мережі та конкуренції каналів

Ще одним критичним фактором стала конкуренція за доступ до мережевого ресурсу. У сценарії з навмисним створенням навантаження (запуск торрент-клієнта, онлайн-стрімінг, синхронізація з хмарними сервісами), спостерігалось суттєве зниження не лише швидкості, а й якості з'єднання — збільшення затримки та нестабільність результатів при повторних тестах.

Було також помічено, що тип стороннього трафіку впливає на результат:

- при потоковому відео (YouTube, Netflix) швидкість падала повільно, але стабільно;
- при скачуванні файлів через P2P — результат знижувався різко, до 15–20 Мб/с, іноді ще нижче;
- фонове оновлення системи (Windows Update) теж суттєво впливало на доступний ресурс.

Це ще раз доводить, що мережевий контекст під час проведення вимірювання є ключовим фактором, який необхідно враховувати при інтерпретації результатів.

Взаємозв'язок між стабільністю сигналу та повторюваністю результатів

У ході тривалих експериментів було виявлено, що:

- у стабільному середовищі результати майже не змінюються між запусками (розбіжність $< 3\%$);
- при нестабільному сигналі або сильному навантаженні мережі похибка могла сягати до 30% між повторними тестами.

Такий ефект спостерігався переважно у бездротових з'єднаннях або при роботі з мережами, де одночасно діяло багато пристроїв. Це підкреслює важливість багатократного тестування у нестабільному середовищі та усереднення результатів для отримання достовірної оцінки.

Поведінка при граничних умовах

Окремо тестувалася робота програми у граничних ситуаціях — наприклад, при дуже короткому інтервалі тестування (2–3 секунди), або навпаки — при тривалому безперервному передаванні (до 60 секунд). У коротких тестах фіксувалася значна варіативність через обмеження кількості переданих пакетів, тоді як довші тести демонстрували стабільніші й репрезентативніші результати.

Крім того, в окремому сценарії була протестована ситуація, коли клієнт або сервер примусово завершували роботу до завершення передачі — програма коректно обробляла таку ситуацію, завершувала вимірювання і виводила результат або повідомлення про помилку, залежно від контексту.

5.3. Порівняльний аналіз ефективності TCP та UDP для різних типів трафіку

Одним з основних завдань дослідження було порівняння ефективності двох найпоширеніших транспортних протоколів — TCP та UDP — у контексті різних типів мережевого трафіку. Обидва протоколи мають принципово різні підходи до передачі даних, що обумовлює їх застосування у конкретних практичних сценаріях.

Для здійснення повноцінного аналізу до клієнт-серверної системи було додано підтримку UDP-протоколу, що дало можливість здійснювати заміри у двох режимах. Крім стандартних показників швидкості, у випадку з UDP були додатково зафіксовані:

- втрата пакетів (packet loss) — як відсоток недоставлених пакетів від загальної кількості надісланих;
- варіабельність затримки (jitter) — як різниця часу доставки між послідовними пакетами.

Ці параметри є критично важливими для оцінки якості трафіку в реальному часі (голос, відео, ігри), де навіть короткі розриви або затримки негативно впливають на користувацький досвід.

Порівняльна таблиця характеристик TCP та UDP:

Характеристика	TCP	UDP
Надійність	Висока — гарантія доставки	Низька — пакети можуть губитися
Контроль доставки	Є (підтвердження, повторна передача)	Відсутній
Середня швидкість	Нижча (через накладні витрати)	Вища (немає очікування ACK)

Затримка (latency)	Вища — через перевірку та контроль	Нижча — швидке пересилання
Втрата пакетів	Мінімальна або нульова	Можлива, особливо при перевантаженні
Придатніс ть	Завантаження файлів, веб-доступ	Відео, VoIP, онлайн-ігри

Результати експериментального тестування

Вимірювання проводились в однакових умовах, при схожих навантаженнях мережі. Було виявлено наступне:

- TCP демонстрував стабільні результати з незначною варіацією. Передача відбувалась гарантовано, без втрат. Проте середня швидкість була на 10–15% нижча, ніж у UDP. Це зумовлено підтвердженням доставки кожного пакета, повторними відправленнями у разі помилки та складнішою логікою.
- UDP забезпечував вищу швидкість, особливо у короткочасних тестах. Але при використанні Wi-Fi або при перевантаженні мережі спостерігались втрати пакетів до 3–7%. Для мультимедійного трафіку це прийнятно, але для точних передач (наприклад, файлів) — неприйнятно.

Особливо показовим був тест на стабільність у реальному часі. Під час паралельного передавання UDP-потoku та TCP-завантаження, UDP-дані доставлялися з низьким пріоритетом, що спричиняло зростання jitter до 40–70 мс, тоді як TCP продовжував працювати стабільно, хоч і знижував швидкість.

Висновки порівняння

Проведене порівняння підтвердило, що:

- TCP підходить для критично важливих задач, де гарантія доставки важливіша за швидкість — наприклад, передача файлів, використання API, робота з базами даних.
- UDP є кращим вибором для задач, що потребують мінімальної затримки і можуть "пробачити" втрату окремих фрагментів — відеоконференції, потокове відео, онлайн-ігри.
- Вибір протоколу повинен здійснюватися з урахуванням типу трафіку, особливостей мережевого середовища та вимог до надійності.

Результати експериментів дозволили визначити не лише технічну ефективність кожного протоколу, а й практичну доцільність їх застосування в залежності від конкретних сценаріїв використання.

Для більш глибокого розуміння переваг і обмежень кожного з протоколів TCP та UDP, у межах експериментальної частини були проведені додаткові серії вимірювань за умов змінного навантаження, використання різного типу мережевого середовища та з урахуванням специфіки типів трафіку, що імітувались у реальному житті.

Тестування на прикладі типових застосувань

Під час порівняння ефективності TCP та UDP було змодельовано три найбільш поширені типи мережевого трафіку:

1. Передача файлів (великі об'єми даних, важлива точність)

Використання TCP показало очікувано високу ефективність — гарантована доставка всіх блоків даних, автоматичне відновлення в разі збоїв, збереження порядку доставки. UDP у цьому випадку виявився непридатним: при найменших втрутах у канал спостерігалися пошкодження даних і втрати фрагментів.

2. Потокове відео (стрімінг)

Тут перевага була на боці UDP. Висока швидкість, низька затримка, мінімальне споживання ресурсів. Навіть при втраті окремих пакетів користувацький досвід не погіршувався. TCP при цьому створював буферизацію, яка підвищувала затримку, а повторні передачі створювали непотрібне навантаження.

3. Інтерактивні ігри / VoIP

У випадках, де затримка критично важлива (ігрові події, голосовий зв'язок), TCP викликав характерні «риивки» — затримки у 100–200 мс через підтвердження доставки. UDP дозволяв підтримувати постійний потік даних зі швидкою передачею, хоча й із періодичними втратами. З погляду суб'єктивного сприйняття — UDP забезпечував кращу плавність.

Поведінка під навантаженням

Окремо тестувалася поведінка TCP та UDP за умов часткової або повної перевантаженості каналу. Результати засвідчили, що:

- TCP автоматично зменшує обсяг переданих даних, уповільнює передачу, чекає на ACK, чим стабілізує з'єднання;
- UDP продовжує відправляти пакети з максимальною швидкістю, що в умовах обмеженої пропускної здатності призводить до зростання втрат (у тестах — до 15–20%).

Це вказує на важливу відмінність: TCP краще пристосований до роботи у погіршених умовах, тоді як UDP або "працює повністю", або "втрачає значну частину інформації", що потребує додаткових механізмів компенсації (наприклад, на рівні застосунку).

Додаткові результати вимірювання

Для більшої наочності у тестуванні були зібрані середні статистичні значення ефективності обох протоколів у різних сценаріях (середня з 5 запусків):

Тип трафіку	ТС P (Мб/с)	UD P (Мб/с)	Втрати UDP (%)	Jitter (мс)
Передача файлів	75.6	85.4	6.2	3–7
Відео 1080p (стрім)	68.2	92.3	4.5	10–20
Онлайн-ігри / VoIP	55.1	88.7	7.9	15–40
Паралельне навантаження	42.8	61.2	14.6	50+

Практичні рекомендації за результатами порівняння

1. TCP рекомендується для:
 - Завантаження/вивантаження файлів;
 - Роботи з базами даних;
 - Вебзастосунків;
 - Ситуацій, де критично важлива цілісність і порядок доставки.
2. UDP рекомендується для:
 - Онлайн-ігор, де швидкість важливіша за точність;
 - Відео- і аудіопотоків, де втрати даних незначні для сприйняття;

- Сценаріїв із великою кількістю клієнтів, що працюють у реальному часі.

Узагальнення

У підсумку можна стверджувати, що ні TCP, ні UDP не є універсальними рішеннями. Кожен із них має переваги у своїй ніші. Проведене експериментальне тестування підтвердило теоретичні відомості й дало змогу зробити висновок: оптимальний вибір протоколу напряму залежить від типу трафіку, характеру задачі, очікуваних параметрів швидкодії, стабільності каналу та вимог до достовірності передачі.

Таким чином, реалізоване програмне забезпечення, яке підтримує обидва протоколи, дозволяє гнучко адаптуватись до потреб користувача та специфіки конкретного середовища.

5.4. Оцінка впливу зовнішніх факторів на швидкість передачі даних

У процесі дослідження було проведено комплексну оцінку впливу зовнішніх факторів, які у реальному середовищі можуть значною мірою змінювати поведінку мережі, незалежно від вибраного протоколу передачі. Такі фактори включають фізичні, технічні та поведінкові аспекти функціонування мережі. Їх урахування є надзвичайно важливим для отримання об'єктивної оцінки продуктивності з'єднання, а також для вірного тлумачення результатів тестування.

1. Якість з'єднання (дротове проти бездротового)

Дослідження показали, що стабільне дротове з'єднання (Ethernet) забезпечує найвищу якість передачі даних. Результати демонстрували не лише високу швидкість (до 95 Мб/с у локальному середовищі), а й відсутність втрат, низьку затримку та повну повторюваність результатів.

На відміну від цього, Wi-Fi з'єднання було схильне до сильних коливань. Навіть незначне переміщення пристрою, поява перешкод або зміна

радіооточення могли спричинити втрати пакетів, зниження швидкості до 30–40% та зростання jitter.

2. Фонові процеси

Особливу увагу було приділено паралельному навантаженню мережі іншими додатками. У ході тестування одночасно з програмою запускались завантаження великих файлів, відеострімінг, синхронізація хмарних сервісів. У результаті фіксувалося зниження швидкості тесту на 20–40%, залежно від характеру навантаження.

При використанні UDP-передачі це також супроводжувалося підвищенням рівня втрат та нестабільністю доставки (помітні коливання в розрахованих значеннях швидкості між запусками).

3. Відстань до маршрутизатора та рівень сигналу

У випадку Wi-Fi-з'єднань було зафіксовано чіткий зв'язок між якістю сигналу та продуктивністю. При зменшенні рівня сигналу спостерігалось:

- зростання середнього ping із 5–10 мс до 50–100 мс;
- збільшення jitter із 10 мс до 70+ мс;
- поява втрат до 5–10% у UDP-режимі;
- зниження загальної швидкості більш ніж у два рази.

4. Тип та якість мережевого обладнання

Було протестовано декілька моделей маршрутизаторів: бюджетні побутові, офісні середнього класу та професійні пристрої з підтримкою QoS.

Результати показали, що дешеві пристрої при високому навантаженні демонстрували значно вищі затримки, схильність до втрат і сповільнення навіть TCP-потоків. У той час як обладнання з пріоритетами обслуговування (QoS) підтримувало стабільність навіть при одночасній роботі з кількома типами трафіку.

5. Протокол передачі як частина загальної системи

Також було встановлено, що вибір протоколу працює у взаємозв'язку з

вищевказаними факторами. Наприклад, у слабкому Wi-Fi середовищі TCP виглядав більш стабільним, хоча й повільним, у той час як UDP дозволяв зберігати темп, але супроводжувався втратою якості.

Висновок:

Ці дослідження підтвердили, що навіть ідеально реалізоване програмне забезпечення не може гарантувати постійну якість вимірювань без урахування зовнішнього контексту. Фізичне середовище, рівень мережевого шуму, якість обладнання, фонові процеси та поведінка користувача — усе це безпосередньо впливає на результати. Саме тому будь-яке тестування мережі має проводитися не ізольовано, а з урахуванням реального середовища використання, і саме для цього реалізоване програмне забезпечення є особливо цінним — воно дозволяє виявляти такі впливи на практиці.

5.5. Інтерпретація отриманих результатів

На основі експериментів, які охоплювали широкий спектр умов, було сформовано низку ключових висновків щодо поведінки протоколів TCP та UDP, а також ефективності реалізованого інструменту.

1. У стабільному дротовому середовищі TCP демонструє найвищу надійність. Його здатність гарантувати доставку кожного пакета та контролювати порядок передачі робить його найкращим вибором для задач, де втрата даних неприпустима — наприклад, передача файлів, робота з базами даних, веб-сервіси.
2. UDP виявився ефективним у динамічному середовищі, де ключовим є мінімальна затримка та постійний потік даних — таких як VoIP, відеоконференції, стрімінг або онлайн-ігри. Програма показала, що UDP-режим забезпечує вищу пропускну здатність і кращу реакцію, але за рахунок меншої надійності.
3. Програмне забезпечення показало високу адаптивність до різних умов мережі. У всіх сценаріях, включаючи перевантаження, нестабільний

сигнал або зміну середовища, система коректно завершувала вимірювання, виводила обґрунтовані результати та надавала користувачу інформативні повідомлення.

4. Фактори, пов'язані з фізичною та логічною організацією мережі, мають вирішальне значення. Відстань до точки доступу, завантаження каналу, тип обладнання та навіть тип кабелю впливали на результати значною мірою. Це підкреслює, що оцінка продуктивності з'єднання — це не лише технічне, а й контекстуальне завдання.

5. Повторюваність результатів підтверджує достовірність розрахунків. Похибка між серіями тестів у стабільному середовищі становила менше 5%, що є прийнятним рівнем для прикладних тестових засобів.

Узагальнення:

Результати свідчать, що розроблене програмне забезпечення є надійним та практичним інструментом, який може використовуватись як у навчальних цілях, так і для реальної оцінки продуктивності мереж у побутових та професійних середовищах. Воно дозволяє не лише вимірювати швидкість, але й виявляти глибинні проблеми мережевої інфраструктури, що є особливо цінним для адміністраторів і розробників.

ВИСНОВКИ

6.1. Узагальнення отриманих результатів

У ході виконання дипломної роботи було створено повнофункціональний клієнт-серверний програмний комплекс для вимірювання швидкості передачі

даних у мережах із використанням протоколів TCP та UDP. Основною метою роботи було забезпечити зручний, ефективний та достовірний спосіб тестування продуктивності мережевого з'єднання в різних умовах — від ідеального локального середовища до реального Wi-Fi-покриття зі змінними параметрами.

На основі проведеного аналізу предметної області було встановлено переваги і недоліки існуючих інструментів, а також обґрунтовано необхідність створення власного рішення, орієнтованого на гнучкість, простоту використання та точність вимірювань. Було вивчено роботу транспортних протоколів TCP та UDP, їхню архітектурну побудову, підходи до управління потоком, підтвердженням доставки та відновленням втрат. Ці знання були застосовані у проектуванні та реалізації програмного забезпечення.

Розроблена система включає:

- інструмент вимірювання швидкості TCP-передачі, з точним розрахунком обсягу переданих даних і часу;
- режим UDP-тестування, що дозволяє виявляти втрати пакетів та затримки;
- графічний інтерфейс, створений з використанням бібліотеки tkinter, для візуалізації результатів у зручному текстовому форматі;
- механізми обробки помилок та збоїв, які гарантують стабільну роботу навіть у складних умовах мережевого середовища.

Проведено експериментальні дослідження, які охоплювали тестування в локальному середовищі, у локальній мережі (LAN), через бездротове з'єднання (Wi-Fi), а також за умов імітації навантаження. Результати цих досліджень підтвердили:

- високу точність вимірювань;
- стійкість програми до впливу зовнішніх чинників;
- можливість застосування інструменту для аналізу ефективності протоколів залежно від типу трафіку.

Система виявилась ефективною не лише для оцінки швидкості передачі, а й для виявлення проблем з мережею, таких як перевантаження, нестабільність Wi-Fi, втрати пакетів тощо. Таким чином, програмне забезпечення виконує не лише діагностичну функцію, а й може використовуватися як засіб аудиту якості з'єднання.

Завдяки простоті інтерфейсу, гнучкості архітектури, точності результатів та адаптивності до різних мережевих умов, розроблена система є універсальним засобом, що може знайти широке застосування:

- у домашніх умовах — для перевірки Wi-Fi або швидкості провайдера;
- у корпоративних мережах — для аналізу продуктивності підключень між вузлами;
- в освітньому процесі — для демонстрації роботи транспортних протоколів і механізмів передачі даних.

Таким чином, реалізоване рішення поєднує дослідницьку, практичну та навчальну цінність, підтверджуючи доцільність і актуальність обраної тематики дипломної роботи.

6.2. Відповідність виконаної роботи поставленим завданням

У процесі виконання дипломної роботи було реалізовано весь комплекс поставлених завдань, які охоплювали як теоретичне обґрунтування проблематики, так і практичну реалізацію та верифікацію розробленого рішення. Кожен етап роботи логічно продовжував попередній, забезпечуючи цілісність підходу та послідовність дослідження.

1. Аналіз існуючих методів вимірювання швидкості

Було вивчено найпоширеніші підходи до тестування мережевої продуктивності, зокрема методи, реалізовані у таких системах як Speedtest, iPerf, NetSpeedMonitor. Було встановлено їхні переваги, обмеження, вимоги до середовища, а також виявлено потребу у створенні гнучкого, локалізованого інструменту з відкритою архітектурою.

2. Дослідження транспортних протоколів TCP і UDP

Теоретичний аналіз дозволив чітко сформулювати принципові відмінності між

TCP та UDP, їхню область застосування, характер поведінки в умовах перевантаження мережі, втрат пакетів, варіації затримки. Це стало основою для обґрунтованого вибору алгоритмів у програмі.

3. Вибір інструментів розробки

Було обрано мову Python, як універсальну, швидку в реалізації і достатньо потужну для мережових задач. Для реалізації GUI використано tkinter, що дозволило створити простий, але функціональний інтерфейс користувача. Основу мережової взаємодії склали бібліотеки socket та threading, які забезпечили низькорівневу гнучкість у керуванні процесами передачі даних.

4. Реалізація архітектури застосунку

Створена клієнт-серверна архітектура передбачає двосторонню взаємодію: клієнт надсилає пакети, сервер приймає й аналізує дані. Обидва компоненти працюють у незалежних потоках, що дозволяє уникати блокування GUI та забезпечує стійкість до збоїв.

5. Проведення всебічного тестування

Програма була протестована у широкому спектрі умов: на одному пристрої, через локальну мережу, по бездротовому з'єднанню, у середовищі з навантаженням. Було досліджено вплив зовнішніх факторів на точність і стабільність вимірювання. Для кожного режиму було зафіксовано показники продуктивності та виявлено типові вузькі місця мережової інфраструктури.

6. Аналіз результатів та їх інтерпретація

Було виконано глибокий аналіз отриманих даних. Порівняльне тестування TCP і UDP дозволило зробити обґрунтовані висновки щодо доцільності використання кожного з протоколів залежно від типу трафіку. Оцінка факторів, таких як тип обладнання, рівень сигналу, фонове навантаження, забезпечила контекст для інтерпретації вимірювань.

Таким чином, усі поставлені цілі були досягнуті:

- Створено робочий інструмент з практичною цінністю;
- Проведено повноцінне технічне і експериментальне обґрунтування;
- Підтверджено відповідність результатів теоретичним припущенням.

Завдяки цьому дипломна робота не лише відповідає формальним вимогам до дослідження, але й має прикладне значення — її результати можуть бути використані в реальних умовах, у сфері мережевої діагностики, оптимізації та навчання.

6.3. Практичні рекомендації щодо використання розробленого програмного забезпечення

Розроблене клієнт-серверне програмне забезпечення може бути успішно застосоване у широкому спектрі завдань, пов'язаних з тестуванням, аналізом і оптимізацією мережевих з'єднань. Його функціональність, простота використання та здатність відображати ключові показники продуктивності дають змогу як звичайним користувачам, так і мережевим адміністраторам ефективно працювати з інструментом у реальних умовах. Нижче наведено конкретні рекомендації щодо практичного використання програми.

1. Для домашніх користувачів та непрофесійної діагностики

- Запускаючи програму на двох пристроях, підключених до однієї Wi-Fi мережі, можна оцінити реальну швидкість обміну в межах локальної мережі, незалежно від інтернет-провайдера.
- Якщо результати значно нижчі за очікувані, це може вказувати на слабе покриття, перешкоди або перевантаження маршрутизатора.
- Особливо зручно використовувати інструмент для визначення оптимального розташування роутера — тест можна запускати з різних кімнат та поверхів.

2. Для технічних спеціалістів і системних адміністраторів

- Програма може служити інструментом оперативної перевірки працездатності сегментів мережі, особливо у великих офісах, де важливо швидко локалізувати зони зниження пропускну здатності або втрат.
- При виявленні проблем із передачею пакетів або нестабільністю зв'язку доцільно використовувати UDP-режим — він дозволяє побачити рівень втрат, коливання затримки (jitter) та оцінити придатність каналу для відео або VoIP-трафіку.

- У TCP-режимі можна оцінювати реальну швидкість передавання даних через кабельні канали, що корисно при налаштуванні серверів, NAS-сховищ, систем резервного копіювання тощо.

3. Для мережевого планування та оптимізації

- Розроблене ПЗ може використовуватись на етапі проєктування мережі: запуск тестів у різних точках дозволяє оцінити баланс навантаження, покриття, вузькі місця та проблеми конфігурації.

- На базі серії вимірювань можна формувати карту пропускну здатності мережі, особливо у складних об'єктах з великою кількістю точок доступу або кабельних сегментів.

4. Для навчальних цілей

- Програма є наочним прикладом роботи з низькорівневою мережею у Python, що робить її придатною для лабораторних занять з комп'ютерних мереж, операційних систем або програмування.

- Використання TCP- та UDP-режимів дозволяє пояснювати студентам різницю між надійною та ненадійною доставкою, між контрольованими та неконтрольованими протоколами, між підтвердженням і передачею без гарантії.

- Можливість запускати тести в режимі реального часу дозволяє виконувати демонстрації змін параметрів мережі, як-от втрата пакета або зміна ширини каналу, і бачити результат негайно.

5. Для використання в службах підтримки

- Інструмент може бути рекомендований користувачам для самостійного первинного тестування швидкості та стабільності при зверненні до служби підтримки.

- За допомогою простої інструкції користувач може зібрати інформацію щодо пропускну здатності або втрат і передати ці дані технічному спеціалісту для подальшої діагностики.

Таким чином, розроблене програмне забезпечення є універсальним та практичним засобом для тестування мережі. Воно може ефективно використовуватись як у побуті, так і в технічному або академічному середовищі, забезпечуючи точні вимірювання, просту взаємодію та можливість адаптації до широкого кола сценаріїв.

6.4. Напрями подальших досліджень

Попри те, що створене програмне забезпечення повністю виконує основні функції з вимірювання швидкості передачі даних у мережі, його архітектура є відкритою до розширення та гнучкою для вдосконалення. Це дозволяє продовжити дослідження у різних напрямках, що зробить систему ще більш функціональною, зручною для користувача та здатною до масштабування. Нижче подано розширене бачення можливих векторів розвитку.

1. Багатоклієнтська підтримка

У поточній реалізації сервер обслуговує лише одного клієнта за один сеанс, що обмежує можливості тестування в умовах реальної багатокористувацької мережі. Розширення функціоналу до багатопоточності на боці сервера дозволить:

- одночасно тестувати передачу з кількох джерел;
- оцінювати вплив паралельних сесій на загальну продуктивність;
- моделювати конкуренцію за канал у корпоративних мережах.

2. Графічна візуалізація результатів

Додавання графіків та візуальних елементів, зокрема динамічної побудови графіка зміни швидкості, коливань jitter чи втрат, значно підвищить інформативність програми. Це дозволить користувачу:

- краще інтерпретувати результати;
- виявляти піки навантаження;
- проводити порівняння між сеансами.

3. Підтримка IPv6 та розширене налаштування з'єднання
Сучасні мережі дедалі активніше переходять на IPv6. Додавання підтримки цього протоколу забезпечить:

- актуальність програми в сучасних мережеских умовах;
- можливість тестувати з'єднання в змішаних середовищах IPv4/IPv6;
- розширення функціоналу GUI для вибору адрес, портів, тайм-аутів та розміру пакетів.

4. Кросплатформеність
На даний момент реалізація орієнтована на Windows. Випуск версій для Linux та macOS забезпечить:

- ширший спектр застосування в професійному середовищі;
- можливість інтеграції в серверну інфраструктуру;
- підвищення надійності та масштабованості рішення.

5. Журналювання та збереження результатів
Впровадження логуювання тестів та експорту до файлів (CSV, JSON, PDF) дозволить:

- вести історію замірів;
- здійснювати порівняльний аналіз у динаміці;
- зручно надавати результати третім сторонам (провайдеру, технічній підтримці).

6. Мобільна версія програми
Адаптація інструменту для Android або iOS дозволить використовувати його:

- у польових умовах (на об'єктах, без доступу до ПК);
- для перевірки Wi-Fi покриття по будівлі;

- як легкий тестовий клієнт для будь-якої мережі за кілька кліків.

7. Інтеграція з системами моніторингу

Програму можна розвивати в бік автоматичного мережевого моніторингу. Зокрема:

- налаштування регулярних тестів за розкладом;
- оповіщення при виявленні аномалій;
- API для інтеграції з системами адміністрування (Zabbix, Nagios, Grafana).

Таким чином, перспективи розвитку розробленої системи охоплюють як технічні, так і прикладні аспекти, включаючи розширення функціоналу, інтерфейсу, масштабованість та адаптацію до мобільного середовища. Подальші дослідження та розробка на основі цієї системи можуть сприяти створенню повноцінного багатофункціонального комплексу для мережевого аудиту, навчання, тестування та автоматичної діагностики.

Завдяки цьому дипломна робота має стійке прикладне значення, а її результати можуть бути використані не лише як демонстраційний приклад, але й як основа для реального продукту, що здатен знайти застосування у професійних IT-середовищах, службах технічної підтримки, освітніх установах і навіть у побутовому використанні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація Python — <https://docs.python.org/3/>
2. Модуль socket в Python — <https://docs.python.org/3/library/socket.html>
3. Модуль threading в Python — <https://docs.python.org/3/library/threading.html>
4. Опис модуля tkinter — <https://docs.python.org/3/library/tkinter.html>
5. Документація бібліотеки Scapy — <https://scapy.readthedocs.io/>
6. Speedtest by Ookla — <https://www.speedtest.net/>
7. Інструмент iPerf — <https://iperf.fr/>
8. NetSpeedMonitor — <https://www.netspeedmonitor.org/>
9. Офіційний сайт Wireshark — <https://www.wireshark.org/>
10. Посібник користувача Wireshark —
https://www.wireshark.org/docs/wsug_html_chunked/
11. Сокет-програмування на Python (GeeksforGeeks) —
<https://www.geeksforgeeks.org/socket-programming-python/>
12. Клієнт-сервер TCP на Python (Real Python) —
<https://realpython.com/python-sockets/>
13. Python-реалізація тесту швидкості —
<https://towardsdatascience.com/network-speed-test-using-python-2c89b0512e3e>
14. Тег socket на Stack Overflow —
<https://stackoverflow.com/questions/tagged/socket>
15. Основи UDP-протоколу (Cloudflare) —
<https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-ud>

p/

- 16.Основи TCP-протоколу (Cloudflare) —
<https://www.cloudflare.com/learning/ddos/glossary/transmission-control-protocol-tcp/>
- 17.RFC 793: Transmission Control Protocol —
<https://datatracker.ietf.org/doc/html/rfc793>
- 18.RFC 768: User Datagram Protocol —
<https://datatracker.ietf.org/doc/html/rfc768>
- 19.RFC 2544: Benchmarking Methodology —
<https://datatracker.ietf.org/doc/html/rfc2544>
- 20.RFC 6349: Framework for TCP Throughput Testing —
<https://datatracker.ietf.org/doc/html/rfc6349>
- 21.WebSockets API (MDN) —
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- 22.Speedtest CLI на Python —
<https://pythonawesome.com/a-tool-to-test-internet-bandwidth-using-speedtest-cli/>
- 23.speedtest-cli на PyPI — <https://pypi.org/project/speedtest-cli/>
- 24.HowToGeek: гід з мереж — <https://www.howtogeek.com/>
- 25.TCP в Linux (man7) — <https://linux.die.net/man/7/tcp>
- 26.Winsock API (Microsoft Docs) —
<https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page>
- 27.Socket-сервер на Python (opensource.com) —
<https://opensource.com/article/19/5/python-socket-programming>

28. Оптимізація мережі в Linux (Arch Wiki) —
https://wiki.archlinux.org/title/Improving_performance
29. Netcat: утиліта для TCP/UDP — <https://www.gnu.org/software/netcat/>
30. GitHub репозиторій iPerf — <https://github.com/esnet/iperf>
31. GitHub репозиторій speedtest-cli — <https://github.com/sivel/speedtest-cli>
32. NetworkWorld — <https://www.networkworld.com/>
33. PacketPushers — <https://www.packetpushers.net/>
34. Python Guide: Networking —
<https://docs.python-guide.org/scenarios/networking/>
35. TCP vs UDP (IBM Docs) —
<https://www.ibm.com/docs/en/linux-on-systems?topic=network-udp-vs-tcp>
36. Освітній портал Educative.io —
<https://www.educative.io/blog/python-socket-programming>
37. Теоретичний огляд TCP і UDP (Baeldung) —
<https://www.baeldung.com/cs/tcp-vs-udp>
38. Різниця TCP та UDP (GeeksforGeeks) —
<https://www.geeksforgeeks.org/differences-between-tcp-and-udp/>
39. UDP клієнт-сервер на Python (TutorialsPoint) —
<https://www.tutorialspoint.com/udp-in-python>
40. PyMOTW: socket — <https://pymotw.com/3/socket/>
41. Модуль time — <https://docs.python.org/3/library/time.html>
42. NetworkCalc — <https://networkcalc.com/>
43. Microsoft Tech Community — <https://techcommunity.microsoft.com/>

44.PacketLife — <https://packetlife.net/>

45.NetworkLessons — <https://networklessons.com/>

ДОДАТКИ

Код програми:

```
import socket
import time
import threading
import tkinter as tk
from tkinter import scrolledtext

BUFFER_SIZE = 1024 * 64 # 64 KB пакети
TEST_DURATION = 10 # Час тесту у секундах
PORT = 5001 # Порт для тесту
STOP_SIGNAL = b"STOP_TEST"

server_thread = None
client_thread = None

def start_server(result_dict):
    """ Сервер приймає трафік і вимірює швидкість прийому. """
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(("127.0.0.1", PORT))
    server_socket.listen(1)
    server_socket.settimeout(TEST_DURATION + 5)

    try:
        conn, addr = server_socket.accept()
        total_data = 0
        start_time = time.time()

        while time.time() - start_time < TEST_DURATION:
            data = conn.recv(BUFFER_SIZE)
            if not data or data == STOP_SIGNAL:
                break
            total_data += len(data)

        elapsed_time = time.time() - start_time
        speed_mbps = (total_data / elapsed_time) / 1024 / 1024 # Мб/с
```

```
    result_dict["server_speed"] = speed_mbps
    log_message(f"Швидкість прийому (сервер): {speed_mbps:.2f}
Мб/с")
```

```
    conn.close()
except socket.timeout:
    result_dict["server_speed"] = 0
    log_message("Сервер завершив роботу через таймаут.")
finally:
    server_socket.close()
```

```
def start_client(result_dict):
    """ Клієнт відправляє дані для вимірювання швидкості передачі. """
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.settimeout(TEST_DURATION + 5)

    try:
        client_socket.connect(("127.0.0.1", PORT))
        data = b"0" * BUFFER_SIZE
        start_time = time.time()
        total_data = 0

        while time.time() - start_time < TEST_DURATION:
            client_socket.send(data)
            total_data += len(data)

        elapsed_time = time.time() - start_time
        speed_mbps = (total_data / elapsed_time) / 1024 / 1024 # Мб/с

        client_socket.send(STOP_SIGNAL)
        result_dict["client_speed"] = speed_mbps
        log_message(f"Швидкість передачі (клієнт): {speed_mbps:.2f} Мб/с")
    except socket.error:
        result_dict["client_speed"] = 0
        log_message("Помилка клієнта: неможливо підключитися.")
    finally:
        client_socket.close()
```

```
def run_speed_test():
```

```

""" Запускає сервер і клієнт у фонових потоках без зависань. """
global server_thread, client_thread
log_message("Запуск тесту швидкості...")

result_dict = {}

server_thread = threading.Thread(target=start_server, args=(result_dict,))
client_thread = threading.Thread(target=start_client, args=(result_dict,))

server_thread.start()
time.sleep(1) # Очікуємо запуск сервера
client_thread.start()

# Очікуємо завершення тесту
root.after(100, check_results, result_dict)

def check_results(result_dict):
    """ Перевіряє завершення тесту без зависання Tkinter. """
    if not server_thread.is_alive() and not client_thread.is_alive():
        log_message("Тест завершено.")
    else:
        root.after(100, check_results, result_dict) # Продовжуємо перевіряти

def stop_processes():
    """ Зупиняє сервер і клієнт, якщо вони ще працюють. """
    global server_thread, client_thread
    if server_thread and server_thread.is_alive():
        server_thread.join(timeout=1)
    if client_thread and client_thread.is_alive():
        client_thread.join(timeout=1)
    server_thread = None
    client_thread = None
    log_message("Тест зупинено.")

def log_message(message):
    """ Виводить повідомлення у вікно GUI. """
    output_text.insert(tk.END, message + "\n")
    output_text.see(tk.END)
    output_text.update()

```

```
if __name__ == "__main__":  
    # === Графічний інтерфейс ===  
    root = tk.Tk()  
    root.title("Тест швидкості передачі даних")  
    root.geometry("600x350")  
  
    output_text = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=70,  
height=15)  
    output_text.pack(pady=10)  
  
    start_button = tk.Button(root, text="Запустити тест",  
command=run_speed_test)  
    start_button.pack()  
  
    stop_button = tk.Button(root, text="Зупинити тест",  
command=stop_processes)  
    stop_button.pack()  
  
    root.mainloop()
```

Приклад виконання

