НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“____”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: Методи підвищення швидкодії клієнт-сервер для real-time додатків на основі модифікованих UDP-протоколів

Виконав (-ла):
студент (-ка) IV курсу, групи ТР-02

ЗАЩИТИНСЬКА Марія Олександрівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
доцент, к.т.н., Ігор КУЗЬМЕНКО _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент: _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *асистент Антон ПАСІЧНЮК* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
“___”_____2024 р.

ЗАВДАННЯ
на дипломну роботу студентці

ЗАЩИТИНСЬКІЙ Марії Олександрівні
(прізвище, ім’я, по батькові)

1. Тема роботи: **Методи підвищення швидкодії клієнт-сервер для real-time додатків на основі модифікованих UDP-протоколів**

Керівник роботи: КУЗЬМЕНКО Ігор Миколайович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “__” _____ 202__ р. № _____

2. Термін подання роботи студентом 07.06.2024 року

3. Вихідні дані до роботи: мова програмування Python, середовище розробки PyCharm, інструмент для симуляції нестабільної мережі clumsy

4. Перелік питань, які необхідно розробити:

1) провести аналіз транспортних протоколів

2) проаналізувати алгоритми розв’язання проблем втрати пакетів та невпорядкованості даних у протоколі UDP

3) розробити модифікацію UDP-протоколу та програмне забезпечення для передачі даних на його основі

4) протестувати розроблене програмне забезпечення для оцінки ефективності

5. Орієнтовний перелік графічного (ілюстративного) матеріалу: схеми, що відображають механізм роботи транспортних протоколів; діаграма, що демонструє алгоритм роботи при втраті пакету; реалізації серверної і клієнтської частини; приклади роботи з програмним продуктом

6. Дата видачі завдання “15” вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи	19.09.2023	
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.24	
8.	Передзахист	03-06.06.24	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-21.06.2024	

Студентка _____ **Марія ЗАЩИТИНСЬКА**
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник _____ **Ігор КУЗЬМЕНКО**
(підпис) (ім’я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 49 сторінках, містить 13 ілюстрацій, 1 таблицю, 1 додаток, 22 джерела в переліку посилань.

Мета роботи – розробка модифікованого UDP-протоколу для підвищення швидкодії та надійності клієнт-серверу для передачі файлів у реальному часі.

Методи та засоби: алгоритми вирішення проблем втрати пакетів та невпорядкованості даних при передачі на основі протоколу UDP, мова програмування Python, інтегроване середовище розробки PyCharm, модуль для роботи з мережевими з'єднаннями socket, стандартні модулі Python.

Результат – клієнт-серверна система для передачі файлів на основі модифікованого UDP-протоколу, який підвищує надійність і продуктивність передачі даних в реальних умовах мережі.

Ключові слова: МОДИФІКОВАНИЙ UDP-ПРОТОКОЛ, ПЕРЕДАЧА ДАНИХ, ШВИДКОДІЯ, НАДІЙНІСТЬ, КЛІЄНТ-СЕРВЕР.

ABSTRACT

The thesis consists of 49 pages, contains 13 illustrations, 1 table, 1 appendix, 22 sources in the list of references.

The purpose of the work is to develop a modified UDP protocol to increase the speed and reliability of the client-server for real-time file transfer.

Methods and tools: algorithms for solving the problems of packet loss and data disorder based on the UDP protocol, the Python programming language, the PyCharm integrated development environment, a module for working with socket network connections, standard Python modules.

The result is a client-server system for file transfer based on a modified UDP protocol that increases the reliability and performance of data transfer in real network conditions

Keywords: MODIFIED UDP-PROTOCOL, DATA TRANSMISSION, SPEED, RELIABILITY, CLIENT-SERVER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Мережевий протокол - набір правил, що дозволяє здійснювати з'єднання і обмін даними між двома і більше включеними в мережі пристроями.

Модель OSI (Open Systems Interconnection Basic Reference Model) – абстрактна мережева модель для комунікацій і розробки мережевих протоколів.

UDP (User Datagram Protocol) – протокол транспортного рівня моделі OSI, який працює без встановлення з’єднання.

TCP (Transmission Control Protocol) – протокол транспортного рівня моделі OSI, який працює зі встановленням з’єднання.

Пакет даних – форматований блок інформації, що передається через комп'ютерну мережу з використанням технології комутації пакетів

Сокет – програмний інтерфейс для забезпечення обміну даними між процесами.

ACK – символ управління передачею, що означає підтвердження отримання пакету (acknowledgement).

NACK – символ управління передачею, що означає помилку в отриманні пакету (negative acknowledgement).

ВСТУП

У сучасному світі інформаційних технологій, швидкість обробки та передачі даних має вирішальне значення для ефективності програмного забезпечення, що вимагає обробки даних в реальному часі. Зокрема, транспортні протоколи, такі як UDP (User Datagram Protocol), часто обирають за їх ефективність у сценаріях, де швидкість важливіша за надійність доставки. Однак, стандартні реалізації UDP не завжди здатні задовольнити потреби додатків реального часу, оскільки вони не забезпечують механізми для вирішення проблем з втратою пакетів та контролем їх впорядкування, що може суттєво впливати на продуктивність додатку.

Актуальність предметної області досліджень щодо підвищення швидкодії клієнт-серверних додатків на основі модифікованих UDP-протоколів визначається зростаючим попитом на real-time додатки у таких сферах, як ігрова індустрія та стрімінгові платформи. Проблеми, що виникають через невпорядкованість або втрати даних, призводять до значних затримок та зниження продуктивності додатків. Розвиток цієї області важливий для забезпечення низької затримки і високої пропускної здатності у спілкуванні між клієнтами і серверами в реальному часі, що стає все більш критичним у сучасному цифровому середовищі.

Предметом дослідження є модифіковані UDP-протоколи, спрямовані на оптимізацію надійності і швидкодії передачі даних для додатків у реальному часі. Об'єктом дослідження є оцінка швидкодії та ефективності цих протоколів у реальних умовах експлуатації real-time додатків, зокрема, в контексті мережевої взаємодії між клієнтами і серверами.

Метою даної дипломної роботи є розробка модифікованого UDP-протоколу для підвищення швидкодії та надійності клієнт-сервер для real-time додатків, щоб оптимізувати передачу даних в умовах, що вимагають високої пропускної здатності та мінімальних затримок.

Для досягнення цієї мети були поставлені такі завдання:

1) аналіз підходів, методів та алгоритмів розв’язання проблем втрати пакетів та невпорядкованості даних у протоколі UDP;

2) аналіз програмних засобів для реалізації програмної системи, що забезпечує надійність і швидкодію передачі даних;

3) розробка модифікованого UDP-протоколу та програмного забезпечення для передачі даних на основі цих модифікацій;

4) тестування розробленого програмного забезпечення для оцінки його ефективності та надійності в реальних умовах експлуатації.

Структура роботи включає перелік умовних позначень, скорочень і термінів, вступ, 4 розділи основної частини, висновки, список використаних джерел, додаток. У розділі 1 проаналізовано проблеми продуктивності систем на основі транспортних протоколів. У розділі 2 проведений огляд алгоритмів підвищення швидкодії та надійності систем на основі модифікованого протоколу UDP. У розділі 3 розглянуто засоби розробки програмного забезпечення. У розділі 4 описано програмну реалізацію системи.

Робота містить 1 додаток, 13 рисунків, 1 таблицю та 22 використаних джерела. Загальний обсяг роботи складає 58 сторінок.

1 АНАЛІЗ ПРОБЛЕМИ ПРОДУКТИВНОСТІ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ ДЛЯ РЕАЛЬНОГО ЧАСУ НА ОСНОВІ ТРАНСПОРТНИХ ПРОТОКОЛІВ

У сучасних клієнт-серверних системах для реального часу, таких як відеоконференції, онлайн-ігри та фінансові транзакції, критично важливими є швидкодія, надійність та низька затримка передачі даних. Традиційні транспортні протоколи, як TCP та UDP, мають свої переваги і недоліки в цьому контексті. У цьому розділі розглянемо прикладну задачу, оглянемо існуючі транспортні протоколи та методи підвищення продуктивності таких систем.

1.1 Постановка прикладної задачі

Задачею даної роботи є розробити та впровадити програмну систему для оптимізації передачі файлів через мережу з використанням модифікованого UDP-протоколу в клієнт-сервері, з метою підвищення швидкодії та ефективності мережевої взаємодії.

Для досягнення поставленої мети у розробці програмної системи необхідно виконати наступні завдання:

- провести аналіз і вибір оптимальних протоколів транспортного рівня та їх модифікацій з урахуванням вимог real-time додатків та особливостей клієнт-серверної архітектури;
- провести дослідження вимог до швидкодії передачі файлів у real-time додатках з використанням модифікованих UDP-протоколів, визначити технічні та функціональні вимоги до програмної системи з урахуванням потреб клієнт-серверного середовища;
- реалізувати модифікований UDP-протокол, який забезпечує оптимізовану передачу файлів через мережу;

− провести комплексне тестування розробленої програмної системи для перевірки її функціональності, стабільності та відповідності вимогам real-time додатків та виконати валідацію системи на реальних умовах мережевої взаємодії.

Розроблена програмна система повинна забезпечувати оптимізовану передачу файлів через мережу з використанням модифікованого UDP-протоколу в real-time клієнт-серверних додатках, забезпечуючи швидкодію, ефективність та надійність мережевої взаємодії.

1.2 Огляд транспортних протоколів

Транспортні протоколи є ключовим компонентом мережевих комунікацій, забезпечуючи передачу даних між клієнтами та серверами. Вони визначають правила, за якими здійснюється передача пакетів даних, а також методи контролю цілісності та доставки інформації. Серед найбільш відомих транспортних протоколів виділяються TCP (Transmission Control Protocol) та UDP (User Datagram Protocol), кожен з яких має свої унікальні характеристики та сфери застосування. У цьому пункті ми розглянемо основні особливості цих протоколів, їхні переваги та недоліки, а також проаналізуємо їх відповідність вимогам клієнт-серверних систем для реального часу.

1.2.1 Аналіз протоколів транспортного рівня

Транспортні протоколи відіграють критичну роль у моделі мережевих комунікацій OSI (Open Systems Interconnection), забезпечуючи надійну передачу даних через мережу на транспортному рівні [2]. Ці протоколи, як TCP (Transmission Control Protocol) і UDP (User Datagram Protocol), необхідні для регулювання обміну даними між кінцевими точками системи, визначаючи механізми контролю за цілісністю, послідовністю і доступністю переданих пакетів.

Основними завданнями транспортного рівня є:

− розбивка повідомлення сеансового рівня на пакети, їхня нумерація;

- буферизація прийнятих пакетів;
- впорядкування пакетів, що прибувають;
- адресація прикладних процесів;
- керування потоком [4].

Транспортні протоколи є ключовим компонентом мережевих комунікацій, забезпечуючи передачу даних між клієнтами та серверами. Вони визначають правила, за якими здійснюється передача пакетів даних, а також методи контролю цілісності та доставки інформації.

Основна відмінність між TCP і UDP полягає в тому, що TCP - це протокол, який базується на з'єднанні, а UDP - без з'єднання. Хоча TCP надійніший, він повільніше передає дані. UDP менш надійний, але працює швидше. Це робить кожен протокол придатним для різних типів передачі даних [5].

TCP працює за допомогою "тристороннього рукостискання" - триетапного процесу, який формує з'єднання між пристроєм і сервером. Завершення цього процесу встановлює безперервне з'єднання, починає передачу пакетів даних через Інтернет, доставляє їх у цілісності та підтверджує доставку.

Технологію роботи TCP можна пояснити трьома кроками (рисунок 1.1):

- клієнтський пристрій, який ініціює передачу даних, надсилає серверу порядковий номер (SYN), з якого має починатися передача пакету даних;
- сервер підтверджує клієнтський SYN і надсилає свій власний SYN-номер (SYN-ACK (SYN-acknowledgment));
- потім клієнт підтверджує (ACK) SYN-ACK сервера, що формує пряме з'єднання і починає передачу даних.

З'єднання між відправником і одержувачем підтримується до тих пір, поки передача не буде успішною. Кожного разу, коли пакет даних надсилається, він вимагає підтвердження від одержувача. Якщо підтвердження не отримано, дані відправляються повторно.

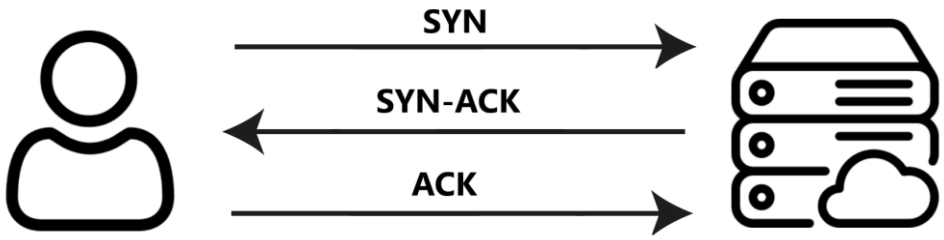

Рисунок 1.1 – Тристоронній процес рукостискання протоколу TCP

Протокол UDP працює шляхом негайної передачі даних одержувачу, який зробив запит на передачу даних, доки передача не буде завершена або припинена. Іноді його називають протоколом "fire-and-forget", UDP надсилає дані одержувачу в довільній послідовності, без підтвердження доставки або перевірки того, чи пакети надійшли за призначенням (рисунок 1.2).

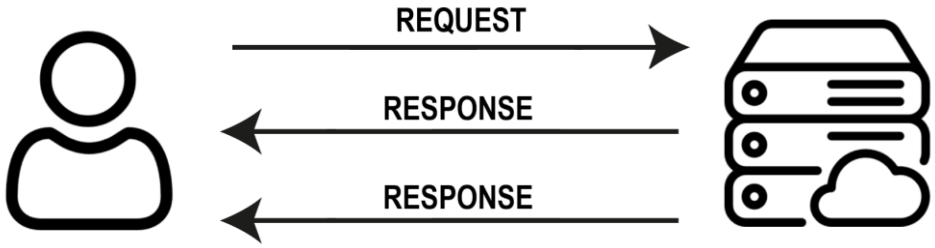

Рисунок 1.2 – Механізм передачі даних за допомогою UDP-протоколу

Відсутність процедур перевірки доставки і повторної передачі робить UDP більш вразливим до втрат пакетів у порівнянні з більш надійним TCP (Transmission Control Protocol). Однак це також означає, що протокол UDP займає менше часу на обробку та передачу даних, що може бути важливим в мережах з високою пропускною здатністю або в умовах, коли об'єм переданих даних дуже великий. Тому, хоча UDP не забезпечує гарантії доставки, його використання залишається виправданим у додатках, які можуть толерувати деякі помилки в обміні даними або потребують максимальної швидкості передачі [12].

У той час як TCP встановлює формальне з'єднання за допомогою угоди "рукостискання" перед відправкою даних. UDP не має на це часу. Він прискорює передачу даних, надсилаючи пакети без жодних домовленостей з одержувачем [17].

Основні відмінності між протоколами UDP і TCP відображені в таблиці 2.1.

Таблиця 1.1 – Порівняльна характеристика протоколів TCP і UDP

Фактор	Протокол TCP	Протокол UDP
Тип з’єднання	Вимагає встановленого з’єднання перед передачею	З’єднання не встановлюється
Послідовність даних	Може впорядковувати дані (надсилає в певному порядку)	Неможливо впорядкувати дані
Перевідправлення даних	Може перенадсилати дані, якщо пакети були втрачені	Повторна передача не відбувається, втрачені пакети неможливо відновити
Доставка	Доставка гарантована	Доставка не гарантована
Перевірка на помилки	Ретельна перевірка помилок гарантує, що дані надійдуть в тому стані як були відправлені	Мінімальна перевірка помилок
Швидкість	Низька, але всі дані гарантовано отримуються	Висока, але є ризик втрати даних

Таким чином, TCP забезпечує надійну та впорядковану передачу даних завдяки механізмам встановлення з'єднання та перевірки помилок. У свою чергу, UDP відрізняється високою швидкістю передачі та мінімальними накладними витратами, що робить його ідеальним для реального часу та додатків, де можлива втрата деяких пакетів.

1.2.2 Переваги та недоліки протоколів UDP та TCP

Продовжуючи аналіз протоколів транспортного рівня, важливо розглянути переваги та недоліки кожного з них. TCP та UDP мають свої унікальні характеристики, які роблять їх корисними для різних типів додатків.

Спочатку розглянемо переваги й недоліки TCP-протоколу. Він є одним з найбільш широко використовуваних протоколів транспортного рівня, що забезпечує надійність. Перевагами цього протоколу є:

- гарантія надійності передачі даних;
- забезпечення механізму перевірки помилок і відновлення пакетів;
- забезпечення контролю потоку.

З іншого боку, TCP також має певні недоліки, які можуть бути критичними для певних додатків:

- можливе сповільнення швидкості мережі;
- відсутність жодних модифікацій з моменту розробки;
- розмір може стати проблемою для невеликих мереж з обмеженими ресурсами [22].

Отже, TCP має свої переваги у забезпечення надійності і контролю передачі даних, але також має певні недолік, які обмежують його ефективність. Тепер перейдемо до аналізу протоколу UDP, який широко відомий своєю швидкодією і низьким рівнем надійності.

Переваги UDP-протоколу включають:

- високу швидкість передачі даних;
- менші затримки, оскільки не потребує встановлення з'єднання;
- простоту структури, що робить легшим управління ним;
- менший розмір пакетів, що зменшує перевантаження мережі;
- підходить для додатків реального часу, де важлива швидкість, а не надійність.

Розглянемо також недоліки UDP-протоколу, оскільки їх потрібно враховувати для додатків, яким необхідна надійність при передачі:

- відсутність гарантії доставки пакетів;

- вразливість до атак та збоїв;
- відсутність механізму перевірки помилок та відновлення пакетів;
- дані можуть надходити в довільному порядку [19].

Отже, TCP та UDP мають свої унікальні переваги та недоліки, які визначають їх придатність для різних типів додатків. TCP забезпечує надійність, контроль потоку та впорядкованість даних, але може сповільнювати мережу через значні накладні витрати. З іншого боку, UDP пропонує високу швидкість і низькі затримки, що робить його ідеальним для додатків реального часу, хоч і без гарантії доставки та перевірки помилок.

1.3 Огляд методів підвищення продуктивності

Ефективна передача даних є ключовим фактором для підтримки високої продуктивності в клієнт-серверних системах, особливо в реальному часі, де швидкість реагування та надійність є критично важливими. З покращенням мережевих технологій і збільшенням вимог до застосунків реального часу, розробники постійно шукають способи оптимізації продуктивності. Це може бути досягнуто за допомогою різноманітних методів та підходів, які допомагають мінімізувати затримки, підвищити швидкість передачі даних, і зменшити навантаження на мережеві ресурси.

Протокол UDP, хоч і простий та швидкий, має свої недоліки, зокрема відсутність гарантії доставки та впорядкованості даних. Удосконалення протоколу UDP можуть включати не лише зміни в самому протоколі, але й використання додаткових стратегій та алгоритмів для підвищення надійності та продуктивності передачі даних. Нижче розглянемо кілька можливих удосконалень UDP та алгоритмів.

1. Додавання механізмів для підвищення надійності – однією з основних модифікацій UDP може бути впровадження механізмів підтвердження отримання пакетів (ACK). Подібний механізм може включати надсилання підтверджень

(ACK/NACK) – одержувач відправляє підтвердження отримання пакетів або повідомляє про втрачені пакети і тайм-аут та повторну передачу – відправник очікує підтвердження протягом певного часу і за потреби повторно відправляє пакети.

2. Контроль порядку пакетів – для додатків, де важлива послідовність доставки даних, можна додати механізми нумерації та впорядкування пакетів. Це включає нумерацію пакетів – додавання унікального ідентифікатора для кожного UDP пакета та буферизацію та впорядкування – одержувач буферизує пакети і розміщує їх у правильному порядку перед обробкою [11].

3. Використання фрагментації та збору пакетів – розділення великих блоків даних на менші пакети може зменшити ймовірність втрати даних та забезпечити кращу адаптацію до умов мережі.

4. Управління потоком та контроль заторів – встановлення механізмів для контролю за швидкістю передачі даних і управлінням потоком може допомогти уникнути перевантаження мережі та забезпечити стабільну передачу даних навіть при високому навантаженні.

Ці удосконалення можуть допомогти зробити протокол UDP більш придатним для використання в різноманітних сценаріях, де потрібна швидка та надійна передача даних. Шляхом впровадження таких механізмів, системи можуть ефективно використовувати протокол UDP, забезпечуючи високу продуктивність та надійність передачі даних, не перевантажуючи мережеві ресурси.

Загалом, поєднання простоти та швидкості протоколу UDP із механізмами покращення надійності та продуктивності може забезпечити ефективну та надійну передачу даних у клієнт-серверних системах реального часу.

2 АЛГОРИТМИ ПІДВИЩЕННЯ ШВИДКОДІЇ ТА НАДІЙНОСТІ МОДИФІКОВАНИХ UDP-ПРОТОКОЛІВ

Для забезпечення ефективної та надійної передачі даних у мережі, модифіковані UDP-протоколи потребують впровадження спеціальних алгоритмів і механізмів. Стандартний UDP-протокол забезпечує високу швидкість передачі, але страждає від недостатньої надійності через відсутність механізмів підтвердження отримання та контролю послідовності пакетів. У цьому розділі ми розглянемо різні алгоритми та механізми, спрямовані на підвищення швидкодії та надійності передачі даних за допомогою модифікованих UDP-протоколів. Ми зосередимося на оптимізації передачі пакетів, розробці механізмів повторної передачі втрачених пакетів, впровадженні контролю послідовності та впорядкування пакетів, а також покращенні безпеки передачі даних. Кожен з цих підходів допоможе зменшити вплив втрат пакетів і підвищити загальну ефективність протоколу.

2.1 Оптимізація UDP-протоколу для передачі пакетів

Оптимізація UDP-протоколів для передачі пакетів є важливим завданням для підвищення ефективності та надійності передачі даних у мережі. Стандартний UDP-протокол забезпечує високу швидкість передачі завдяки своїй простоті, але при цьому не гарантує доставку пакетів і не забезпечує механізмів для обробки втрат даних. Тому необхідно впроваджувати різні оптимізаційні стратегії для вирішення цих проблем.

Фрагментація великих даних дозволяє розділяти великі файли на менші фрагменти, що уникати проблем з передачею великих пакетів і зменшує ймовірність втрат. З іншого боку, об'єднання декількох невеликих пакетів в один великий знижує накладні витрати на передачу кожного окремого пакета і підвищує ефективність передачі. Це особливо корисно в умовах, коли передачі дрібних пакетів можуть значно знижувати загальну продуктивність мережі.

Динамічне регулювання швидкості передачі є ще однією важливою стратегією оптимізації. Воно полягає в адаптації швидкості передачі даних залежно від умов мережі, таких як затримка і рівень втрат пакетів. Це допомагає уникнути перевантаження мережі та підвищити надійність передачі. Віконний механізм контролю, який обмежує кількість надісланих пакетів у мережі, також сприяє оптимізації потоку даних і покращенню загальної продуктивності.

Зменшення накладних витрат може бути досягнуто за рахунок оптимізації заголовків пакетів. Використання мінімальних заголовків дозволяє зменшити накладні витрати і підвищити ефективність передачі. Наприклад, можна впровадити стислий формат заголовків.

Оптимізація буферизації на стороні клієнта і сервера є ще одним важливим аспектом. Ефективні алгоритми буферизації допомагають керувати чергами пакетів і запобігати перевантаженню буферів. Впровадження механізмів пріоритизації для критично важливих пакетів дозволяє забезпечити більш швидку і надійну доставку важливих даних, що є особливо актуальним для систем з високими вимогами до якості обслуговування [20].

Застосування цих оптимізаційних стратегій дозволяє значно підвищити ефективність і надійність передачі даних за допомогою UDP-протоколу, зберігаючи при цьому його основні переваги, такі як низька затримка і висока пропускна здатність.

2.2 Механізм повторної передачі втрачених пакетів

Механізм повторної передачі втрачених пакетів є критично важливим для забезпечення надійної передачі даних в мережах, що використовують протокол UDP. UDP, як відомо, не забезпечує гарантії доставки пакетів і не включає вбудовані механізми відновлення втрачених даних. Тому для підвищення надійності передачі необхідно реалізувати додаткові механізми, які дозволяють ефективно виявляти втрати пакетів і ініціювати їх повторну передачу.

1. Виявлення втрат пакетів – основою механізму повторної передачі є виявлення втрат пакетів. У традиційному UDP-протоколі немає засобів для перевірки цілісності або порядку отриманих пакетів, тому необхідно впровадити додаткові методи для відстеження та виявлення втрачених пакетів. Один з найпоширеніших методів – використання порядкових номерів пакетів. Кожен пакет, відправлений від клієнта до сервера, містить унікальний порядковий номер, який дозволяє серверу визначити, чи були пропущені якісь пакети.

Коли сервер отримує пакет, він перевіряє його порядковий номер і порівнює з очікуваним. Якщо номер отриманого пакета не відповідає очікуваному, це свідчить про втрату одного або кількох пакетів. Наприклад, якщо сервер отримав пакет з номером 3, а наступний пакет має номер 5, сервер визначає, що пакет з номером 4 був втрачений.

2. Запити на повторну передачу – після виявлення втрат пакетів сервер ініціює процес повторної передачі. Це здійснюється шляхом відправлення спеціального повідомлення-запиту (наприклад, NACK4) до клієнта з вказівкою, який пакет було втрачено і потребує повторної передачі. У відповідь на запит клієнт відправляє втрачений пакет знову.

Цей механізм може бути реалізований кількома способами. Один з них - це індивідуальні запити для кожного втраченого пакета. Такий підхід забезпечує високу точність, але може бути неефективним у випадку значних втрат пакетів, оскільки кожен запит і відповідь додають накладні витрати.

Інший підхід – це групові запити, коли сервер запитує повторну передачу кількох втрачених пакетів одночасно. Це зменшує кількість запитів і відповідей, підвищуючи загальну ефективність процесу відновлення даних.

3. Підтвердження отримання – для забезпечення надійності передачі даних також важливо впровадити механізми підтвердження отримання пакетів (ACK). Коли сервер отримує і обробляє пакет, він надсилає клієнту підтвердження отримання цього пакета. Це дозволяє клієнту відстежувати, які пакети були успішно доставлені, а які потребують повторної передачі.

Підтвердження можуть бути реалізовані як для кожного окремого пакета, так і для групи пакетів. Вибір методу залежить від конкретних умов мережі і вимог до ефективності. Наприклад, для мереж з високим рівнем втрат більш доцільно використовувати підтвердження для кожного пакета, щоб уникнути необхідності повторної передачі значних обсягів даних. Водночас для мереж з низьким рівнем втрат можна використовувати групові підтвердження, що знижує накладні витрати на передачу.

4. Тайм-аут – ще один важливий компонент механізму повторної передачі. Якщо сервер не отримує підтвердження від клієнта протягом певного часу, він припускає, що пакет був втрачений, і повторно відправляє його. Визначення оптимального часу тайм-ауту є критично важливим для ефективної роботи цього механізму. Занадто короткий час тайм-ауту може призвести до надмірної повторної передачі пакетів, що збільшить навантаження на мережу. Занадто довгий час, навпаки, знизить швидкодію передачі даних.

5. Віконний механізм – є ще одним ефективним способом управління потоком даних і повторної передачі втрачених пакетів. Цей механізм дозволяє передавати кілька пакетів без очікування підтвердження отримання кожного з них, що підвищує ефективність передачі. Віконний механізм також дозволяє відстежувати які пакети були підтверджені, а які – ні, що спрощує процес повторної передачі.

6. Інтеграція з іншими механізмами надійності – механізм повторної передачі втрачених пакетів часто інтегрується з іншими механізмами надійності, такими як контроль цілісності даних і попереджувальне кодування помилок. Використання контрольних сум дозволяє виявляти помилки у переданих даних і ініціювати повторну передачу тільки пошкоджених пакетів, що підвищує ефективність процесу. Попереджувальне кодування помилок додає до пакетів додаткові дані, що дозволяють відновлювати втрачені або пошкоджені пакети без необхідності їх повторної передачі, знижуючи тим самим навантаження на мережу.

На рисунку 2.1 відображено алгоритм передачі втраченого пакету у вигляді діаграми послідовності.

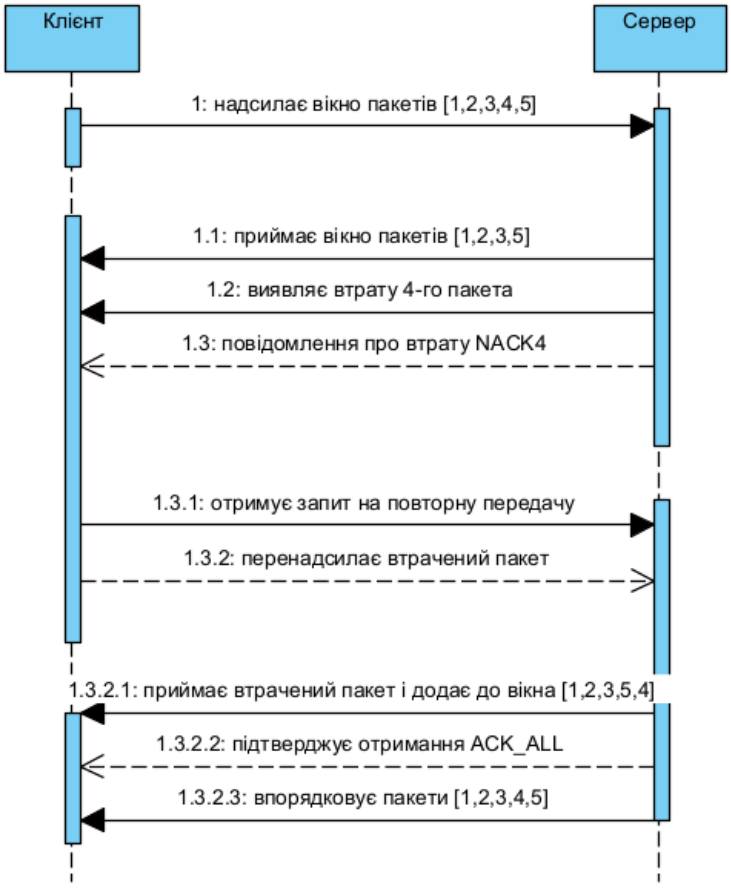

Рисунок 2.1 – Діаграма послідовності для повторної передачі втраченого пакету

Механізм повторної передачі втрачених пакетів є необхідною складовою для забезпечення надійності передачі даних у мережах, що використовують UDP-протокол. Впровадження порядкових номерів пакетів, запитів на повторну передачу, підтвердження отримання, тайм-аутів і віконного механізму дозволяє значно підвищити ефективність і надійність передачі даних. Ці механізми забезпечують можливість відновлення втрачених пакетів, знижують ймовірність втрат даних і покращують загальну продуктивність мережі, роблячи UDP-протокол більш придатним для використання в сучасних мережах з високими вимогами до надійності та швидкодії. Крім того, впровадження повторної передачі втрачених пакетів не лише підвищує надійність, а й дозволяє більш ефективно використовувати ресурси мережі. Це є особливо важливим для додатків реального часу, де невеликі втрати можуть призвести до погіршення якості зв’язку.

2.3 Механізм контролю послідовності та впорядкування пакетів

UDP-протокол, хоч і є швидким та ефективним у передачі даних, не гарантує послідовність або цілісність пакетів. Це означає, що пакети можуть приходити до місця призначення в неправильному порядку або бути втраченими. Для забезпечення надійної передачі даних необхідно впровадити механізми контролю послідовності та впорядкування пакетів. Ці механізми забезпечують правильний порядок пакетів, дозволяють виявляти відсутні або дубльовані пакети і відновлювати коректну послідовність даних.

1. Впровадження порядкових номерів – як і для втрат пакетів, одним з основних методів контролю послідовності є використання порядкових номерів пакетів. Кожен пакет, що відправляється від клієнта до сервера, містить унікальний порядковий номер, який дозволяє серверу визначити правильний порядок пакетів. При отриманні пакета сервер перевіряє його порядковий номер і порівнює з очікуваним. Якщо номер отриманого пакета не відповідає очікуваному, це свідчить про втрату одного або кількох пакетів або про їх отримання в неправильному порядку.

2. Буферизація та перевпорядкування пакетів - для забезпечення правильної послідовності пакетів сервер використовує буфер для тимчасового зберігання пакетів, що прийшли в неправильному порядку. Коли сервер отримує пакет з номером, що не відповідає очікуваному, він зберігає цей пакет у буфері до отримання всіх попередніх пакетів. Наприклад, якщо сервер очікує отримати пакет з номером 5, а отримує пакет з номером 6, він зберігає пакет 6 у буфері, поки не отримає пакет 5. Після отримання пакета 5 сервер вилучає пакет 6 з буфера і обробляє його.

Буферизація дозволяє серверу впорядковувати пакети і забезпечувати правильний порядок даних. Однак ефективність цього механізму залежить від розміру буфера і складності алгоритму перевпорядкування. Занадто малий буфер може призвести до втрат пакетів, тоді як занадто великий буфер збільшить затримки в обробці даних.

3. Використання віконного механізму – цей механізм контролю передачі також може бути використаний для забезпечення послідовності та впорядкування пакетів. Він дозволяє клієнту відправляти кілька пакетів без очікування підтвердження отримання кожного з них, що підвищує ефективність передачі. Клієнт відправляє пакети у "вікні" певного розміру, а сервер підтверджує отримання вікна пакетів після їх обробки.

Віконний механізм дозволяє серверу перевіряти послідовність отриманих пакетів у вікні і виявляти пропущені пакети. Якщо сервер виявляє пропущені пакети, він може запитати їх повторну передачу, використовуючи механізм підтвердження отримання (ACK) та негативного підтвердження (NACK).

Механізм контролю послідовності та впорядкування пакетів значно підвищує надійність передачі даних у мережах, що використовують UDP-протокол. Він дозволяє виявляти втрати, забезпечує правильний порядок даних і зменшує ймовірність виникнення помилок при передачі. Однак цей механізм має деякі недоліки, які необхідно враховувати при його впровадженні.

Перш за все, використання порядкових номерів і буферизація пакетів додають додаткові накладні витрати на передачу та обробку даних. Це може збільшити затримки в передачі та обробці даних, особливо в умовах високого навантаження на мережу. Крім того, ефективність буферизації залежить від розміру буфера і складності алгоритму перевпорядкування, що може вимагати додаткових ресурсів сервера.

Механізм контролю послідовності та впорядкування пакетів є необхідною складовою для забезпечення надійної передачі даних у мережах, що використовують UDP-протокол. Використання порядкових номерів, буферизація та перевпорядкування пакетів, виявлення та обробка втрачених пакетів, а також застосування віконного механізму дозволяють значно підвищити надійність і ефективність передачі даних. Хоча цей механізм додає додаткові накладні витрати, він забезпечує надійність передачі даних у мережах з високими вимогами до цілісності та послідовності даних.

2.4 Механізм покращення безпеки

Безпека є одним з ключових аспектів передачі даних у мережах, особливо коли йдеться про використання UDP-протоколу, який за своєю природою не забезпечує жодного рівня захисту. У цьому підрозділі розглянемо механізми покращення безпеки передачі даних за допомогою модифікованого UDP-протоколу.

1. Автентифікація клієнта – є першочерговим заходом для забезпечення безпеки передачі даних. Вона дозволяє серверу перевіряти, чи дійсно запит на передачу даних походить від авторизованого клієнта. Процес автентифікації складається з кількох етапів. Перш ніж клієнт зможе надсилати файли, він повинен ініціювати з'єднання з сервером. Це передбачає відправлення запиту на встановлення з'єднання, який містить початкову інформацію про клієнта. Сервер зберігає IP-адресу клієнта та номер порту у своєму списку клієнтів. Це дозволяє серверу відстежувати, які клієнти встановили з'єднання і мати можливість перевіряти їхні подальші запити. Коли сервер отримує пакет з запитом на надсилання файлу, він перевіряє інформацію про клієнта у своєму списку. Якщо інформація збігається, сервер надсилає клієнту підтвердження автентифікації (ACK). У випадку, якщо інформація не збігається, сервер відхиляє запит, надсилаючи клієнту негативну відповідь (NACK) і відмовляється приймати файл.

Автентифікація клієнта забезпечує захист від неавторизованих доступів і зловмисних атак, таких як підміна адрес або відправка даних неавторизованими клієнтами. Це дозволяє гарантувати, що тільки довірені клієнти можуть передавати дані до сервера.

2. Перевірка розміру вікна – є ще одним важливим заходом безпеки, який допомагає запобігти різним атакам і забезпечити стабільну передачу даних. Розмір вікна визначає кількість пакетів, які можуть бути одночасно відправлені від клієнта до сервера без підтвердження отримання кожного окремо. Узгодження розміру вікна між клієнтом і сервером є критичним для забезпечення ефективної та безпечної передачі даних.

Після встановлення з'єднання клієнт відправляє серверу інформацію про розмір вікна. Це дозволяє серверу знати, скільки пакетів клієнт планує відправляти без очікування підтвердження. Далі сервер перевіряє отриману інформацію про розмір вікна і порівнює її з власними налаштуваннями. Якщо розмір вікна клієнта збігається з очікуваним сервером розміром, сервер надсилає підтвердження (ACK) і дозволяє передачу даних. Якщо розмір вікна, отриманий від клієнта, не збігається з очікуваним, сервер відхиляє запит, надсилаючи негативну відповідь (NACK). Це запобігає некоректній передачі даних і потенційним атакам, що використовують маніпуляції з розміром вікна для перевантаження сервера або викликання нестабільної роботи системи.

Перевірка розміру вікна забезпечує контроль за потоком даних і допомагає підтримувати ефективну і безпечну передачу даних у мережі. Вона також дозволяє уникати ситуацій, коли некоректно налаштовані клієнти можуть порушувати роботу сервера.

Впровадження механізмів покращення безпеки, таких як автентифікація клієнта та перевірка розміру вікна, значно підвищує надійність і безпеку передачі даних за допомогою UDP-протоколу. Автентифікація клієнта дозволяє захистити систему від неавторизованих доступів, а перевірка розміру вікна забезпечує контроль за потоком даних. Разом ці механізми створюють комплексний підхід до забезпечення безпеки передачі даних, що є критичним для сучасних мережевих систем.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних систем вимагає знань та використання різних технологій, які працюють у взаємодії для створення функціональних рішень. У цьому розділі описані ключові технології, які були використані для розробки клієнт-сервера для підвищення швидкодії передачі файлів через мережу з використанням модифікованих UDP-протоколів.

3.1 Мова програмування Python

Python є однією з найпопулярніших мов програмування, відомою своєю простотою, зручністю для читання коду та потужними бібліотеками. Її широке використання в різних галузях, від веб-розробки до аналізу даних та машинного навчання, робить Python ідеальним вибором для розробки різноманітних програмних рішень, включаючи мережеві [1] .

Для розробки клієнт-серверної архітектури, яка забезпечує високу швидкодію передачі файлів через мережу з використанням модифікованих UDP-протоколів, Python має кілька важливих переваг.

1. Простота та зрозумілість синтаксису – Python має простий та зрозумілий синтаксис, що дозволяє швидко розробляти та відлагоджувати код. Це особливо важливо при розробці мережевих застосунків, де швидка ітерація та тестування є критичними [21].

2. Широкий вибір модулів та бібліотек – Python має багатий набір стандартних бібліотек, таких як socket, які надають базові інструменти для роботи з мережевими протоколами [5]. Для роботи з UDP-протоколами, Python забезпечує легкий доступ до низькорівневих мережевих функцій.

3. Кросплатформеність – Python є кросплатформеною мовою, що означає можливість запуску коду на різних операційних системах без змін. Це спрощує процес розробки та тестування мережевих застосунків у різних середовищах.

4. Інструменти для тестування та налагодження – Python підтримує різні інструменти для тестування, наприклад ‘unittest’, ‘pytest’, що дозволяють зробити процес тестування автоматизованим. Це є важливим для мережевих застосунків, де потрібно гарантувати надійність та стійкість до помилок.

5. Спільнота та документація – також Python має активну спільноту розробників, що забезпечує доступ до великої кількості ресурсів, таких як документація, курси, форуми, відео-пояснення. Це дозволяє легко та швидко знаходити рішення для проблем, які виникають під час розробки.

6. Гнучкість та розширюваність – завдяки гнучкості та можливості інтеграції з іншими мовами та технологіями, Python дає можливість легко додавати нові функціональні можливості до проєкту. Наприклад, можна використати модулі іншої мови програмування для критично важливих частин коду, щоб підвищити продуктивність додатку.

У контексті розробки клієнт-серверного застосунку для передачі файлів через мережу з використанням модифікованих UDP-протоколів, Python надає всі необхідні засоби для реалізації ефективного та надійного рішення. Завдяки своїй гнучкості та широким можливостям, Python дозволяє швидко адаптуватися до змін та вимог проекту, забезпечуючи при цьому високу продуктивність та стабільність роботи застосунку.

3.2 Модулі та бібліотеки

Для реалізації клієнт-серверної архітектури з підвищеною швидкодією передачі файлів через мережу з використанням модифікованих UDP-протоколів у Python використовуються кілька стандартних модулів та бібліотек. Кожен з них забезпечує специфічну функціональність, необхідну для розробки та оптимізації мережевих застосунків.

Модуль socket є основним інструментом для роботи з мережевими з’єднаннями. Він надає інтерфейс для створення та керування сокетами, що

дозволяє відправляти та отримувати дані через мережу [15]. У контексті UDP-протоколів socket використовується для:

- створення UDP-сокетів;
- встановлення зв'язку між клієнтом та сервером;
- відправки та отримання пакетів даних.

Модуль time надає функції для роботи з часом, що є важливим для синхронізації процесів, вимірювання затримок та реалізації тайм-аутів у мережевих з'єднаннях [18]. Використовується для:

- вимірювання часу передачі даних;
- встановлення тайм-аутів на очікування відповідей.

Модуль os забезпечує інтерфейс для взаємодії з операційною системою і забезпечує портативний спосіб використання залежних від операційної системи функцій [10]. Він використовується для роботи з файловою системою, керування процесами та ін. У нашому контексті os використовується для:

- читання та запису файлів;
- отримання інформації про файлову систему.

Модуль select надає функції для здійснення неблокуючого вводу/виводу. Він дозволяє здійснювати одночасне очікування подій на декількох сокетах, що є критичним для асинхронної роботи з мережевими з'єднаннями [14].

Модуль itertools містить функції для створення ітераторів, що дозволяють ефективно обробляти великі обсяги даних. У нашому проекті він використовується для об'єднання кількох послідовностей [7].

Модуль operator надає функції для виконання стандартних операцій над об'єктами і експортує набір ефективних функцій, що відповідають внутрішнім операторам Python. Він використовується для покращення продуктивності коду шляхом заміни звичайної функції на оператор [9].

Модуль hashlib використовується для створення хешів, що є важливим для забезпечення цілісності та безпеки даних. Хешування використовується для перевірки цілісності файлів після передачі [6].

Ці модулі та бібліотеки забезпечують широкий спектр функціональності, необхідної для ефективної розробки, тестування та експлуатації клієнт-серверної системи для швидкої передачі файлів через мережу. Кожен з них відіграє важливу роль у створенні надійного та продуктивного програмного забезпечення.

3.3 Інтегроване середовище розробки IDE

При розробці програмного забезпечення важливо використовувати ефективні інструменти, що забезпечують зручність написання, відлагодження та тестування коду. Одним з таких інструментів є інтегроване середовище розробки (IDE) PyCharm.

PyCharm — це потужне інтегроване середовище розробки, розроблене компанією JetBrains, яке спеціалізується на Python. PyCharm надає розробникам широкий набір інструментів для підвищення продуктивності та якості коду [8]. Для розробки клієнт-серверної архітектури Pycharm має кілька важливих переваг і особливостей.

1. Зручний редактор коду – PyCharm забезпечує інтелектуальне автозаповнення коду, підсвічування синтаксису, автоматичне форматування та інші функції, які допомагають швидко і точно писати код.

2. Відлагодження – вбудований відладчик дозволяє зупиняти виконання програми на точках зупинки, переглядати значення змінних, виконувати покроковий прохід коду і швидко знаходити та виправляти помилки.

3. Інтеграція з системами контролю версій – PyCharm підтримує інтеграцію з Git та іншими системами контролю версій, що полегшує керування кодом і спільну роботу над проектами.

4. Навігація по коду – PyCharm надає розширені можливості для навігації по коду, включаючи швидкий перехід до визначення функцій, класів та змінних, а також пошук і заміну по всьому проекту.

5. Плагіни та розширення – завдяки широкому вибору плагінів, PyCharm можна легко налаштувати під конкретні потреби розробника, додаючи нові функціональності та інтеграції.

6. Тестування – PyCharm підтримує різноманітні фреймворки, призначені для тестування. Інтегрований інструмент призначений для запуску тестів дає можливість автоматизовано виконувати тестування та аналізувати отримані результати, що оптимізує якість роботи.

7. Рефакторинг – PyCharm пропонує велику кількість інструментів для рефакторингу, таких як перейменування змінних, витягнення методів і класів, оптимізацію імпортів. Це допомагає підтримувати чистоту, структуру і зрозумілість коду.

8. Робота з базами даних – є вбудована підтримка роботи з базами даних, що до зволяє розробникам підключатися до різноманітних систем керування базами даних, виконувати SQL-запити та керувати базами даних з PyCharm. Це полегшує роботу з базами даних і дає можливість використовувати меншу кількість сторонніх інструментів.

9. Інтеграція з хмарними сервісами – PyCharm підтримує інтеграцію з хмарними додатками, наприклад Google Cloud, Microsoft Azure, що дозволяє керувати власними додатками в хмарі та легко їх відкривати.

Також існують і інші популярні середовища розробки на Python, такі як Visual Studio Code, Spyder, Jupyter Notebook. VS Code (Visual Studio Code) відомий великою кількістю доступних розширень, легкістю в інтерфейсі та можливістю налаштувати під різні потреби розробників. Також VS Code є універсальним інструментом для розробки, оскільки підтримує велику кількість мов програмування.

Spyder більш спеціалізований для наукових обчислень і має популярність серед інженерів і дослідників. Він надає інтеграцію з інструментами з потужним інтерфейсом для виконання обчислень у реальному часі. Також Spyder включає функції необхідні для аналізу даних і наукових досліджень, наприклад аналіз результатів або інтерактивне відлагодження.

Jupyter Notebook більш спеціалізується на обчисленнях, аналізі даних і машинному навчанні. Розробники можуть створювати документи з живим кодом, рівняннями, візуалізацією і текстовими поясненнями. Тому він є ідеальним для дослідницької роботи, тому що можна наочно переглядати результати.

У порівнянні з іншими середовищами розробки, PyCharm найбільше підходить для розробників, оскільки він глибоко інтегрується з Python і має велику кількість інструментів для професійної розробки. Також він включає відлагодження, тестування і роботу з базами даних, що робить його хорошим вибором навіть для більших проєктів.

PyCharm є відмінним вибором для розробників Python завдяки своїм багатофункціональним можливостям та підтримці сучасних технологій розробки. Ця IDE забезпечує інтеграцію з численними інструментами та фреймворками, що робить процес розробки більш ефективним і продуктивним. Завдяки інтуїтивно зрозумілому інтерфейсу та розширеним можливостям навігації по коду, PyCharm дозволяє розробникам легко управляти складними проектами та зосереджуватись на вирішенні важливих завдань. Крім того, підтримка численних плагінів і налаштувань дозволяє налаштувати середовище розробки під індивідуальні потреби кожного розробника, забезпечуючи комфортну роботу з будь-якими проектами на Python.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі детально описано процес реалізації клієнт-серверної системи для передачі файлів з використанням модифікованого UDP-протоколу. Наведено архітектуру системи, реалізацію серверної та клієнтської частин.

4.1 Архітектура системи

У цьому пункті описується архітектура клієнт-серверної системи, що була розроблена для ефективної передачі файлів за допомогою модифікованого UDP протоколу. Система складається з клієнтської та серверної частин, кожна з яких має свої функції та взаємодіє одна з одною для забезпечення надійного обміну даними. Нижче наведено детальний огляд клієнт-серверної архітектури, аналізу протоколу та основних компонентів системи.

4.1.1 Огляд клієнт-серверної архітектури

Клієнт-серверна архітектура базується на взаємодії між двома основними компонентами: клієнтом і сервером. Така архітектура дозволяє розподілити завдання між двома учасниками, що сприяє оптимізації використання ресурсів і забезпечує надійну передачу даних. На діаграмі прецедентів відображено основні сценарії взаємодії між клієнтом і сервером (рисунок 4.1).

Клієнтська частина відповідає за кілька ключових завдань. По-перше, вона ініціює з'єднання з сервером. Для цього клієнт створює UDP-сокет і надсилає запит на встановлення з'єднання (CONN_REQ). Після отримання підтвердження від сервера (ACK), клієнт готовий розпочати передачу файлів.

Другим важливим завданням клієнтської частини є розбиття файлу на пакети. Це здійснюється шляхом читання файлу частинами розміром 50 000 байт (значення може варіюватися до 65 535 байтів, що є максимальним розміром UDP-пакету).

Кожен фрагмент даних містить у собі інформацію про порядковий номер пакету, що дозволяє серверу правильно збирати файл у відповідному порядку.

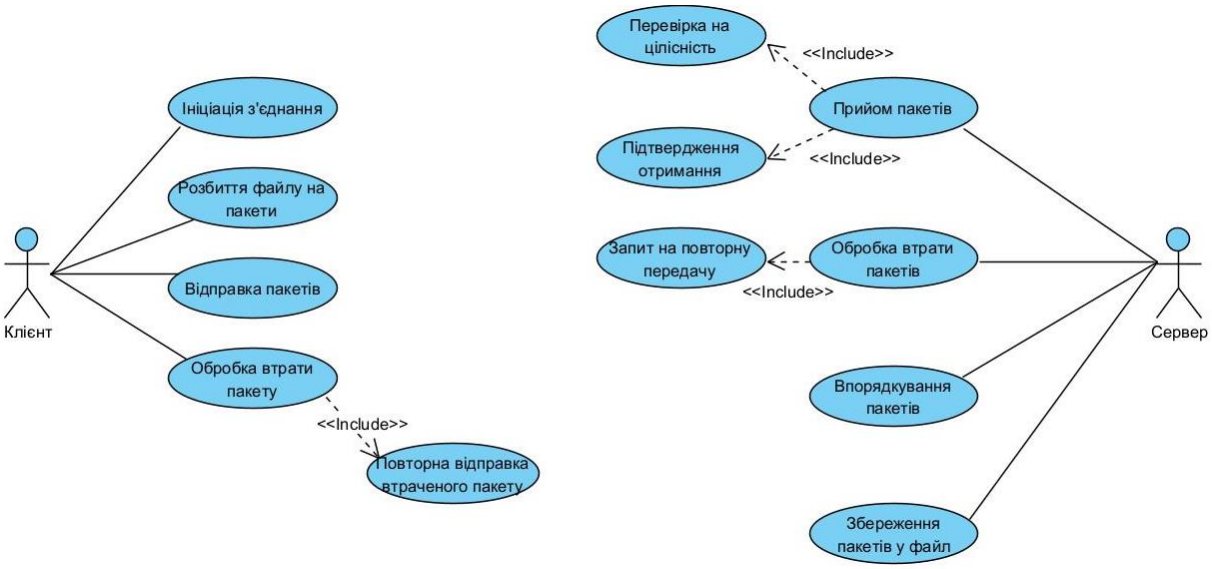

Рисунок 4.1 – Діаграма прецедентів роботи програми

Клієнт також відповідає за відправку цих пакетів на сервер. Пакети надсилаються вікнами, де розмір вікна визначає кількість пакетів, що можуть бути відправлені без очікування підтвердження. Це підвищує ефективність передачі, знижуючи час затримки.

Ще одним важливим завданням клієнта є обробка підтверджень отримання пакетів (ACK) від сервера. Якщо пакет втрачається або отримання не підтверджується у встановлений час, клієнт повторно відправляє відповідний пакет. Такий механізм повторної передачі забезпечує надійність передачі даних навіть у разі втрат пакетів.

Серверна частина системи також виконує кілька важливих функцій. Після отримання запиту на з'єднання від клієнта, сервер підтверджує з'єднання, додаючи клієнта до списку авторизованих клієнтів. Це запобігає несанкціонованому доступу та передачі даних від невідомих джерел.

Основною функцією сервера є прийом пакетів даних від клієнта. Сервер отримує пакети, перевіряє їх на коректність та цілісність і підтверджує отримання

кожного пакету або групи пакетів. Сервер також обробляє можливі втрати даних, надсилаючи клієнту запити на повторну передачу втрачених пакетів (NACK).

Крім того, сервер відповідає за обробку даних у випадку зміни порядку пакетів. Якщо пакети надходять не в тому порядку, в якому були відправлені, сервер сортує їх за індексом перед збереженням у файл. Це дозволяє відновити початковий файл навіть у випадку змішування порядку доставки пакетів.

Таким чином, клієнт-серверна архітектура, реалізована у цій системі, дозволяє ефективно передавати файли через мережу, забезпечуючи високу швидкість, надійність та коректність передачі даних.

4.1.2 Модифікації UDP-протоколу

Для забезпечення надійної та ефективної передачі файлів через мережу, стандартний UDP протокол був модифікований. Нижче розглянемо основні модифікації, впроваджені в нашій системі.

1. Контроль послідовності пакетів – клієнт разом із самим пакетом надсилає номер цього пакета. Це дозволяє серверу розпізнавати невпорядковані або відсутні пакети. Кожен пакет має унікальний номер, що дозволяє серверу ідентифікувати його місце у загальній послідовності. Такий механізм допомагає уникнути плутанини під час збору файлів і гарантує, що всі частини файлу будуть зібрані у правильному порядку.

2. Повторна передача втрачених пакетів – якщо сервер виявляє, що певні пакети були втрачені або не були отримані вчасно, він надсилає клієнту запит на повторну передачу одного або більше відсутніх пакетів. Це дозволяє мінімізувати втрати даних і забезпечити повну передачу файлу.

3. Впорядкування пакетів – пакети, які надходять на сервер, за потреби впорядковуються після отримання всіх пакетів у заданому вікні. Сервер зберігає всі отримані пакети у тимчасовому буфері, поки не отримає всі необхідні пакети з даного вікна. Після цього він сортує їх за номерами і збирає у правильному порядку, що дозволяє відновити початковий файл без втрат даних.

4. Автентифікація клієнта – клієнт спочатку встановлює з’єднання з сервером, перш ніж він зможе надсилати файли. Під час першого підключення сервер зберігає IP-адресу клієнта та номер порту в списку авторизованих клієнтів. Коли сервер отримує пакет з запитом на надсилання файлу, він перевіряє ці дані зі своїм списком клієнтів. Якщо збіг знайдено, сервер надсилає пакет ACK для підтвердження автентифікації. Якщо збігу немає, сервер відхиляє запит пакетом NACK і відмовляється приймати файл. Це гарантує, що тільки авторизовані клієнти можуть надсилати файли.

5. Узгодження розміру вікна – сервер гарантує, що розмір вікна на сервері збігається з розміром вікна клієнта. Це узгодження є критично важливим для забезпечення ефективної передачі даних і уникнення перевантаження мережі. Якщо розмір вікна не узгоджений, це може призвести до втрат пакетів або затримок у передачі.

Всі ці модифікації спрямовані на те, щоб забезпечити високу якість передачі даних і зробити систему більш стійкою до різних мережевих проблем, таких як втрати пакетів, затримки або несанкціоновані доступи. Завдяки цим покращенням, система може ефективно передавати файли навіть у нестабільних або насичених мережах, забезпечуючи при цьому швидкодію, безпеку та надійність передачі даних.

4.2 Реалізація серверної частини

Реалізація серверної частини включає кілька ключових етапів, які забезпечують ефективне отримання, обробку та управління файлами, що передаються від клієнтів. Повну реалізацію можна переглянути в додатку А.

Спочатку на сервері ініціалізується UDP-сокет для прослуховування вхідних запитів від клієнтів. Також цей сокет прив'язується до порту, на якому сервер буде приймати пакети.

Далі відбувається очікування на повідомлення від клієнтів. Коли сервер отримує запит на підключення, він вносить IP-адресу і порт цього клієнта в список клієнтів. Цей список служить для ідентифікації активних клієнтів та управління їх з'єднаннями. Потім при спробі клієнта надіслати файл, сервер перевіряє наявність клієнта в списку і тільки потім надає дозвіл на отримання пакетів. Це дозволяє забезпечити безпеку та контроль доступу, запобігаючи несанкціонованим спробам передачі даних.

Потім відбувається перевірка розміру вікна на збіг з розміром вікна користувача. Процедура перевірки розміру вікна відбувається наступним чином: сервер отримує від клієнта повідомлення із зазначенням розміру вікна, який клієнт планує використовувати для передачі даних. Після отримання цього значення сервер порівнює його з власним розміром вікна.

Якщо розмір вікна, зазначений клієнтом, не відповідає розміру вікна на сервері, сервер надсилає клієнту повідомлення про невідповідність (NACK) і розриває з'єднання. Це дозволяє уникнути потенційних проблем, пов'язаних з неправильною передачею даних через несумісність розмірів вікон.

Основним моментом у процесі передачі даних є перевірка на наявність втрачених пакетів та повідомлення клієнту про необхідність повторної передачі цих пакетів, якщо вони були втрачені. Сервер постійно контролює послідовність отриманих пакетів і в разі виявлення пропусків у пакетах вікна генерує запит на повторну передачу відсутніх даних. Це дозволяє забезпечити цілісність переданих файлів і мінімізувати ризик втрати важливої інформації.

Після успішного прийому всіх пакетів і завершення передачі файлу сервер підтверджує клієнту про завершення процесу. Це досягається шляхом відправки повідомлення про успішне завершення передачі ("ACK_ALL"), що інформує клієнта про те, що всі дані були отримані правильно. У разі успішного завершення всіх перевірок файл зберігається в папці проєкту і стає доступним для подальшої обробки або використання.

Таким чином, реалізація серверної частини забезпечує надійний та ефективний механізм передачі файлів між клієнтом і сервером, враховуючи

можливі проблеми з втратою даних та невпорядкованістю пакетів. Кожен етап процесу оптимізований для досягнення максимального рівня продуктивності та надійності системи.

4.3 Реалізація клієнтської частини

Реалізація клієнтської частини включає кілька кроків, необхідних для ініціації з'єднання з сервером, розбиття файлу на пакети, відправки цих пакетів та обробки підтверджень отримання (ACK).

Перший етап реалізації клієнтської частини передбачає створення UDP-сокета, який використовується для зв'язку з сервером. UDP-сокет дозволяє клієнту надсилати та отримувати пакети без встановлення постійного з'єднання.

Після створення сокета клієнт ініціює з'єднання з сервером, надсилаючи запит на підключення. Сервер у відповідь надсилає підтвердження (ACK), якщо з'єднання встановлено успішно. Якщо з'єднання не вдається встановити, клієнт отримує відповідне повідомлення. Цей етап гарантує, що сервер готовий приймати дані, перш ніж клієнт почне передачу файлу.

Наступний крок – підготовка файлу до передачі. Файл розбивається на менші пакети для полегшення передачі та обробки. Це необхідно для забезпечення того, що великі файли можуть бути передані через мережу з обмеженнями на розмір пакетів. Кожен пакет буде мати свій унікальний номер, що дозволить серверу контролювати послідовність пакетів і виявляти втрати.

Після визначення номеру пакета відбувається перевірка, чи є цей пакет останнім у файлі. Якщо пакет є останнім, до повідомлення з номером пакету додається позначка "F" (фінальний пакет). Це інформує сервер про те, що більше пакетів не буде, і передача файлу завершена. Якщо пакет не є останнім, до повідомлення додається позначка "N" (не фінальний пакет), що вказує на наявність наступних пакетів. Цей механізм допомагає серверу коректно обробляти потік даних і забезпечувати цілісність файлу.

Після надсилання пакетів клієнт чекає на підтвердження від сервера. Якщо сервер повідомляє про втрату пакетів (NACK), клієнт повторно надсилає втрачені пакети. Це забезпечує надійність передачі, оскільки клієнт постійно контролює процес і повторює надсилання втрачених даних до отримання підтвердження від сервера. Клієнт завершує передачу файлу лише після отримання повідомлення від сервера про успішне завершення передачі.

Спочатку отримується порядковий номер втраченого пакета, далі обчислюється номер цього пакета в межах вікна і за цим індексом пересилається на сервер втрачений пакет.

Ці етапи забезпечують повний цикл передачі файлу від клієнта до сервера, включаючи необхідні перевірки, обробку помилок та повторну передачу втрачених пакетів, що дозволяє досягти надійної та ефективної передачі файлів через мережу. Клієнт контролює кожен етап процесу, від ініціації з'єднання до завершення передачі, забезпечуючи високу якість та надійність передачі даних. Це гарантує, що всі пакети будуть доставлені в правильному порядку та без втрат, навіть у випадку несприятливих мережевих умов.

4.4 Демонстрація роботи проєкту

В цьому підрозділі розглядається практична демонстрація роботи проєкту. Основною вимогою до проєкту було вдосконалення і модифікація UDP-протоколу. Тому було вирішено зробити консольний інтерфейс, щоб розбирати і бачити всі складності в передачі і отриманні пакетів та перевірку їхнього впорядкування з умовами нестабільної мережі.

Спочатку бачимо повідомлення що сервер був запущений на локальній адресі та номер порту, де успішно відбулась ініціалізація роботи сервера. Далі отримано запит на підключення клієнта у вигляді повідомлення “CONN_REQ”, яке було підтверджено. Після автентифікації клієнта, сервер отримав запит на передачу

файлу video.mp4. І після підтвердження ідентичності розмірів вікна в сервера і клієнта, була розпочата передача пакетів (рисунок 4.2).

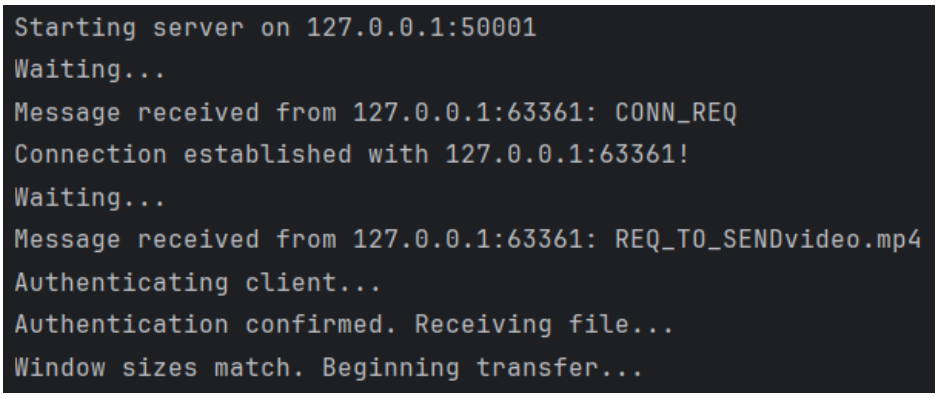

Рисунок 4.2 – Початок роботи сервера до передачі пакетів

Далі бачимо повідомлення про втрату пакетів, що демонструють здатність сервера виявляти втрати та ініціювати механізм повторної передачі, що є ключовим для забезпечення цілісності даних. Після перенадсилання клієнтом втрачених пакетів, перевірки сервером наявності всіх пакетів у вікні, також відбулося їхнє сортування і перевірка правильного порядку (рисунок 4.3).

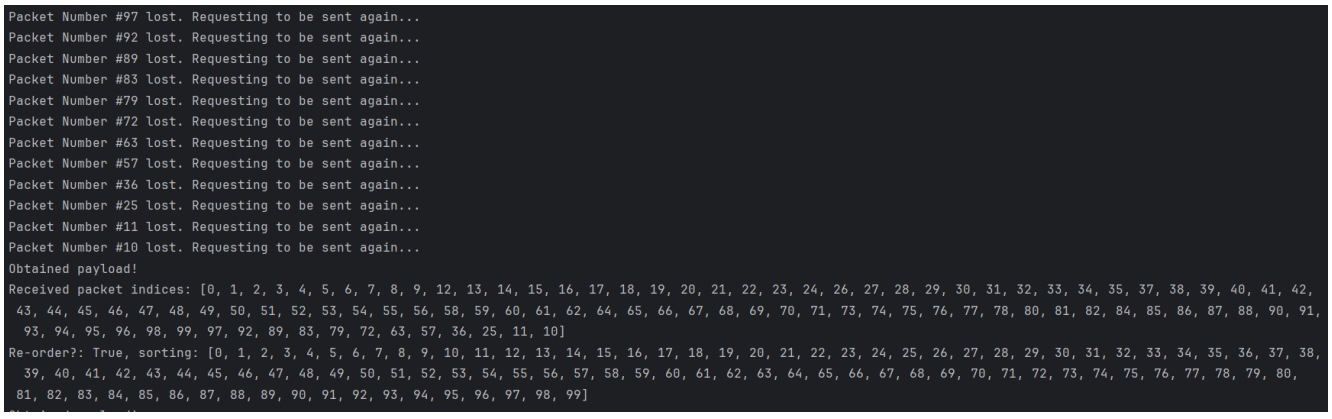

Рисунок 4.3 – Отримання пакетів та їх сортування

Після того як цей етап відбувся з усіма вікнами пакетів, всі пакети було отримані та впорядковані, отримуємо повідомлення про зберігання файлу, його

назву та час витрачений на отримання файлу та збереження. Також було закрито сокет зв’язку між клієнтом і сервером (рисунок 4.4).

```
Saving file. Please wait...
File received_video.mp4 saved successfully!
Closing file...
Час витрачений на передачу файлу: 7.80 сек
Closing server...
```

Рисунок 4.4 – Збереження файлу та закриття сокета

Можемо побачити, що отриманий файл має такий самий розмір і довжину як і надісланий (рисунок 4.5).

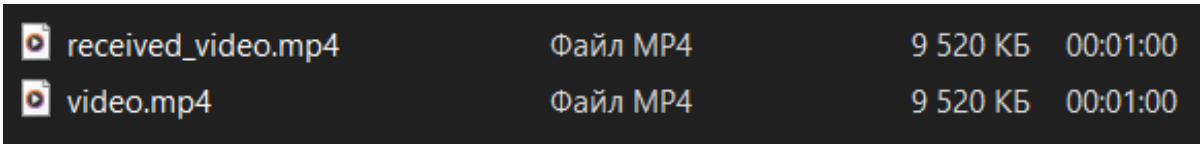

Рисунок 4.5 – Вихідний і отриманий файли

Оскільки потрібно перевірити це ретельніше, було розроблено додаткову функцію для перевірки ідентичності отриманого файлу та вихідного. Ця функція використовує хеш-функцію для обчислення хеш-значення кожного з файлів та порівнює їхні результати. На рисунку 4.6 можемо побачити результати перевірки на ідентичність даних у двох файлах.

```
Файли 'video.mp4' і 'received_video.mp4' ідентичні.
```

Рисунок 4.6 – Результат перевірки ідентичності файлів

Перевірка підтверджує, що сервер виконує свої функції, виявляє та відновлює втрачені пакети, перевіряє їхній порядок та забезпечує надійну передачу пакетів в умовах нестабільного з’єднання. Ця функціональність і надійність системи робить її придатною для застосування в реальних умовах.

4.5 Порівняння швидкодії клієнт-серверів на основі протоколів TCP, стандартного та модифікованого UDP

Порівняння швидкодії клієнт-серверів на основі стандартного і модифікованого UDP та TCP-протоколів є ключовим аспектом для розуміння ефективності різних підходів до передачі даних у мережі. TCP відомий своєю надійністю, але може мати проблеми з продуктивністю і швидкодією у мережах з високим рівнем втрат через механізми контролю потоку та підтвердження отримання пакетів. Стандартний UDP не має механізмів забезпечення надійності, що призводить до втрати пакетів без жодного способу відновлення втрачених даних. Створений модифікований UDP з додатковими механізмами для забезпечення надійності передачі даних може запропонувати альтернативу з потенційно кращою продуктивністю в таких умовах. Для того, щоб дізнатись чи є модифікований протокол ефективнішим за TCP і стандартний UDP було проведено тестування з передачею файлів розміром 10 Мб і 50 Мб з різними відсотками втрат пакетів.

На графіку для файлу розміром 10 Мб (рисунок 4.7) можна побачити, що при 10% втрат пакетів модифікований UDP протокол демонструє гірші результати, витрачаючи приблизно 12 секунд на передачу файлу, тоді як TCP протокол витрачає близько 6 секунд.

При 20% втрат пакетів час передачі для модифікованого UDP збільшується до приблизно 16 секунд, але все стає меншим, ніж у TCP, який витрачає близько 18 секунд. Це вказує на те, що модифікований UDP протокол ефективніше працює зі збільшенням втрат пакетів, ніж TCP.

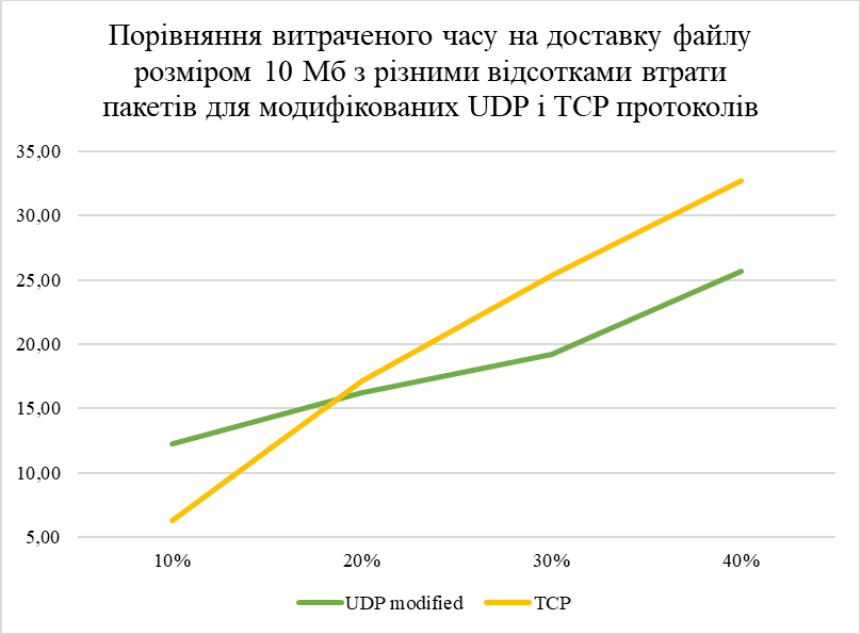

Рисунок 4.7 – Графік порівняння швидкодії модифікованого UDP та TCP протоколів при передачі файлу розміром 10 Мб з різною втратою пакетів

При 30% і 40% втрат пакетів модифікований UDP протокол демонструє поступове збільшення часу до 25 секунд, тоді як TCP значно зростає до 32 секунд. Це підкреслює нижчу ефективність TCP у складних мережевих умовах.

Однак, слід зазначити, що при дуже високих відсотках втрат (40% і більше) обидва протоколи демонструють значне зниження швидкодії. Незважаючи на переваги модифікованого UDP над TCP в таких умовах, час передачі файлів все одно значно зростає, що вказує на обмеження ефективності обох протоколів у надзвичайно високовтратних мережах. Це свідчить про те, що навіть з оптимізаціями, жоден з протоколів не є повністю стійким до великих втрат даних, і можуть бути необхідні додаткові механізми для покращення продуктивності в таких складних мережевих умовах.

Стандартний UDP при будь-яких втратах взагалі не зберігав файл належним чином, оскільки не знав про втрату пакетів, що призводило до неповного або пошкодженого файлу і незавершеності передачі.

На графіку для файлу розміром 50 Мб (рисунок 4.8), при 10% втрат пакетів, TCP протокол демонструє кращі результати, витрачаючи приблизно 11 секунд на передачу, в той час як модифікований UDP витрачає близько 39 секунд.

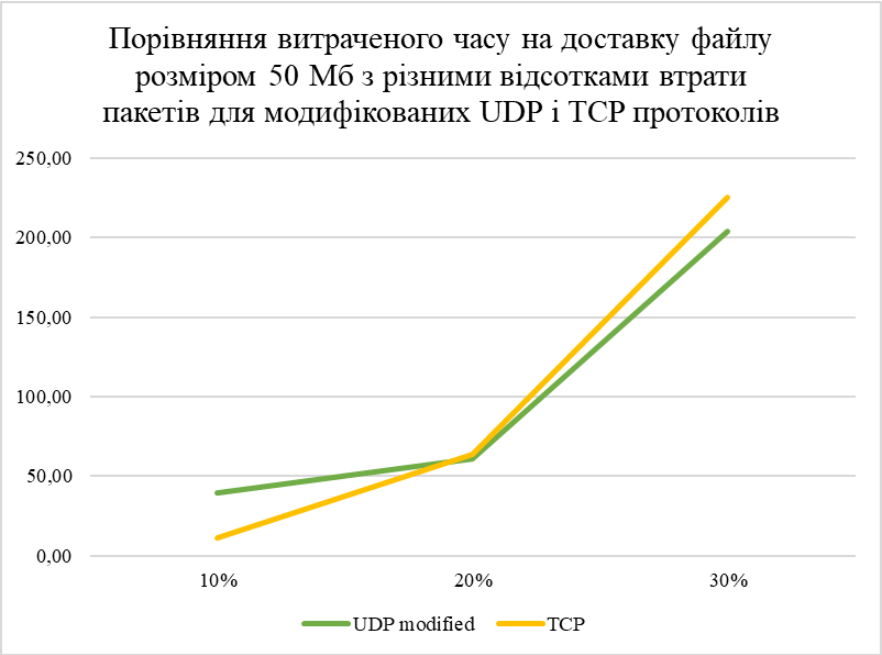

Рисунок 4.8 – Графік порівняння швидкодії модифікованого UDP та TCP протоколів при передачі файлу розміром 50 Мб з різною втратою пакетів

При 20% втрат пакетів модифікований UDP протокол витрачає близько 61 секунди, що все ще менше, ніж час, витрачений TCP (63 секунди). Відмінність між протоколами зменшується, вказуючи на зростаючий вплив втрат пакетів на продуктивність обох протоколів.

При 30% втрат пакетів час передачі для модифікованого UDP протоколу значно зростає до приблизно 204 секунд, тоді як TCP витрачає ще більше часу – близько 225 секунд. Це демонструє значну чутливість обох протоколів до високих рівнів втрат, але модифікований UDP все ще менш чутливий до втрат порівняно з TCP.

Тим не менш, слід зазначити, що обидва протоколи демонструють значне зниження швидкодії при втраті пакетів понад 30%. Незважаючи на те, що модифікований UDP перевершує TCP у цих умовах, час передачі файлів значно

зростає. Це свідчить про те, що обидва протоколи неефективні в мережах з високими втратами даних.

Як висновок, експерименти показали, що модифікований UDP протокол працює краще в умовах низьких і середніх втрат пакетів для файлів малого розміру (10 Мб). Для більших файлів (50 Мб) протокол TCP працює краще при низьких втратах 10%, але поступається модифікованому UDP при збільшенні втрат до 20%. Обидва протоколи демонструють значне зниження продуктивності при втрат 30%.

Таким чином, модифікований UDP може бути кращим для використання в мережах з середнім і низьким рівнем втрат пакетів для малих файлів, тоді як TCP-протокол може бути кращим для стабільних мереж з мінімальними втратами. Стандартний UDP придатний лише для мереж з ідеальними умовами без втрат.

4.6 Порівняння залежності швидкодії клієнт-серверу на основі модифікованого UDP-протоколу від розмірів вікна і пакету

Для того, щоб оцінити вплив різних налаштувань UDP-протоколу на швидкодію, було проведено тестування із зміною таких параметрів, як розмір вікна і розмір пакету. Тести проводилися за різних відсотків втрат пакетів, що дозволило визначити оптимальні конфігурації для кожного з сценаріїв. На основі цих експериментів були створені графіки, які демонструють залежність часу передачі даних від кількості пакетів у вікні та від розміру пакету за умов 10% та 30% втрат.

На першому графіку зображена залежність часу передачі даних від кількості пакетів у вікні при граничних допустимих рівнях втрат пакетів (рисунок 4.9). По осі X відкладено кількість пакетів у вікні (від 10 до 100), а по осі Y – час передачі в секундах. При 10% втрат пакетів час передачі складає приблизно 20 секунд при 10 пакетах у вікні. Зі збільшенням кількості пакетів у вікні до 25, час передачі зменшується до приблизно 18 секунд. Подальше збільшення кількості пакетів до 50 та 100 супроводжується незначним зменшенням часу передачі, що стабілізується на рівні приблизно 15 секунд.

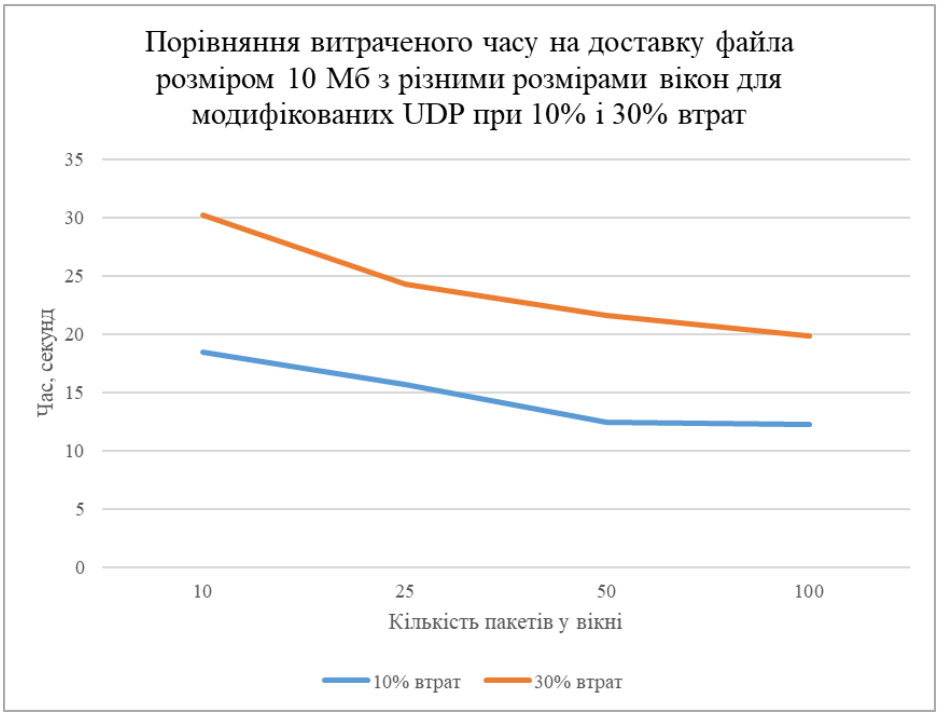

Рисунок 4.9 – Графік залежності швидкодії модифікованого UDP-протоколу від розміру вікна з різною втратою пакетів

При 30% втрат пакетів при 10 пакетах у вікні час передачі становить близько 30 секунд. При збільшенні кількості пакетів у вікні до 25, час передачі зменшується до приблизно 25 секунд. Подальше збільшення кількості пакетів до 50 і 100 приводить до зменшення часу передачі, який стабілізується на рівні приблизно 20 секунд. Тобто, оптимальний розмір вікна при втратах 10-30% буде від 50 до 100 пакетів.

На другому графіку зображена залежність часу передачі даних від розміру пакету при різних рівнях втрат пакетів (рисунок 4.10). По осі X відкладено розмір пакету у кілобайтах (від 1 до 55), а по осі Y – час передачі в секундах. При 10% втрат пакетів при розмірі пакету 1 кілобайт час передачі складає приблизно 20 секунд. Зі збільшенням розміру пакету до 10 кілобайт час передачі зменшується до приблизно 15 секунд. Подальше збільшення розміру пакету до 20, 50 і 55 кілобайт супроводжується незначними коливаннями, але в цілому час передачі зменшується до рівня приблизно 12 секунд.

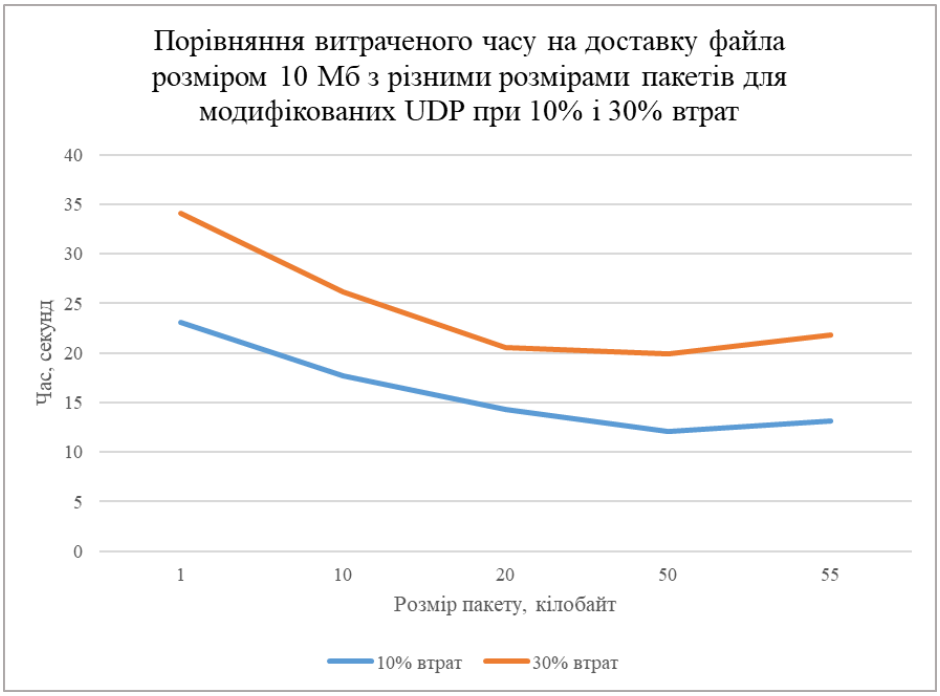

Рисунок 4.10 – Графік залежності швидкодії модифікованого UDP-протоколу від розміру пакета з різною втратою пакетів

При 30% втрат пакетів при розмірі пакету 1 кілобайт час передачі становить близько 35 секунд. При збільшенні розміру пакету до 10 кілобайт час передачі зменшується до приблизно 25 секунд. Подальше збільшення розміру пакету до 20 і 50 кілобайт приводить до зниження часу передачі до приблизно 20 секунд. При збільшенні розміру пакету до 55 кілобайт час передачі зростає до приблизно 22 секунд.

З аналізу графіків можна зробити висновки, що більша кількість пакетів у вікні (50-100 пакетів) дозволяє ефективніше використовувати пропускну здатність мережі навіть при високих втратах. Також, зі збільшенням розміру пакету час передачі зменшується для обох рівнів втрат до певного моменту (приблизно 20-50 кілобайт), після чого при великих розмірах пакету (понад 50 кілобайт) час передачі може зрости, особливо при високих втратах (30%). Це свідчить про те, що оптимальне налаштування розміру вікна та розміру пакету може значно покращити швидкодію UDP-протоколу в умовах різних рівнів втрат пакетів.

ВИСНОВКИ

1. На основі проведеного аналізу сучасних підходів, методів і алгоритмів розв’язання проблем втрат пакетів та невпорядкованості даних у протоколі UDP було обґрунтовано використання механізмів повторної передачі втрачених пакетів, контролю послідовності та впорядкованості, віконного механізму і покращення безпеки. Ці методи дозволяють підвищити надійність передачі даних у реальному часі, знизити ймовірність втрат і покращити загальну продуктивність мережі.

2. На основі аналізу програмних засобів було обґрунтовано використання мови програмування Python, стандартних модулів та бібліотек Python та інтегрованого середовища розробки PyCharm для реалізації програмної системи. Python було обрано завдяки його простоті, зручності читання коду і широкому набору бібліотек і модулів для роботи з мережевими протоколами. PyCharm забезпечує потужні інструменти для відлагодження, інтеграцію з системами контролю версій та розширені можливості навігації по коду, що допомагає ефективності розробки і тестуванню програмного забезпечення.

3. Було розроблено модифікований UDP-протокол, який включає механізми для підвищення надійності і продуктивності передачі даних, такі як повторна передача втрачених пакетів, контроль послідовності пакетів, автентифікація клієнта і віконний механізм. Цей протокол був реалізований у програмному забезпечення для передачі даних між клієнтом і сервером, яке забезпечує високу продуктивність та стабільність під час передачі. Створена система підвищує ефективність мережевої взаємодії , особливо в реальному часі.

4. Комплексне тестування розробленого програмного забезпечення підтвердило його функціональність, стабільність і відповідність вимогам для real-time додатків. Результати тестування показали, що система ефективно працює в нестабільній мережі, демонструючи високу швидкодію у порівнянні з TCP-протоколом при низьких і середніх рівнях втрат пакетів. Це свідчить про те, що модифікований UDP-протокол є придатним для використання у різних мережевих умовах і забезпечує оптимальні результати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Васильєв О. М. Програмування мовою Python : навч. кн. Тернопіль : Богдан, 2021. 504 с.

2. Braun T. Internet protocols for multimedia communications. I. IPng-the foundation of Internet protocols. IEEE Multimedia. 1997. Vol. 4, no. 3. P. 85–90. URL: https://doi.org/10.1109/93.621586 (дата звернення: 27.05.2024).

3. Davie B. S., Peterson L. L. Computer networks: a systems approach. Elsevier Science & Technology Books, 2007. 848 p.

4. Dhas Y. J., Jeyanthi P. A Review on Internet of Things Protocol and Service Oriented Middleware. 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 4–6 April 2019. 2019. URL: https://doi.org/10.1109/iccsp.2019.8698088 (дата звернення: 01.06.2024).

5. Goerzen J., Rhodes B. Foundations of Python Network Programming. 3rd ed. Apress, 2014. 409 p.

6. hashlib – Secure hashes and message digests. Python documentation. URL: https://docs.python.org/3/library/hashlib.html (дата звернення: 17.05.2024).

7. itertools – Functions creating iterators for efficient looping. Python documentation. URL: https://docs.python.org/uk/3/library/itertools.html (дата звернення: 17.05.2024).

8. JetBrains. PyCharm: the Python IDE for data science and web development. JetBrains. URL: https://www.jetbrains.com/pycharm/ (дата звернення: 17.05.2024).

9. operator – Standard operators as functions. Python documentation. URL: https://docs.python.org/uk/3/library/operator.html (дата звернення: 27.05.2024).

10. os – Miscellaneous operating system interfaces. Python documentation. URL: https://docs.python.org/uk/3/library/os.html (дата звернення: 27.05.2024).

11. Packet Reordering in the Era of 6G: Techniques, Challenges, and Applications / J. Lin et al. Electronics. 2023. Vol. 12, no. 14. P. 3023. URL: https://doi.org/10.3390/electronics12143023 (дата звернення: 27.05.2024).

12. Protocols for reliable data transport in space internet / R. Wang et al. IEEE Communications Surveys & Tutorials. 2009. Vol. 11, no. 2. P. 21–32. URL: https://doi.org/10.1109/surv.2009.090203 (дата звернення: 01.06.2024).

13. Python | PyCharm. PyCharm Help. URL: https://www.jetbrains.com/help/pycharm/python.html (дата звернення: 17.05.2024).

14. select – Очікування завершення введення/виведення. Python documentation. URL: https://docs.python.org/uk/3.11/library/select.html (дата звернення: 17.05.2024).

15. socket – Мережевий інтерфейс низького рівня. 3.12.3 Documentation. URL: https://docs.python.org/uk/3.9/library/socket.html (дата звернення: 17.05.2024).

16. Standardized Protocol Stack for the Internet of (Important) Things / M. R. Palattella et al. IEEE Communications Surveys & Tutorials. 2013. Vol. 15, no. 3. P. 1389–1406. URL: https://doi.org/10.1109/surv.2012.111412.00158 (дата звернення: 01.06.2024).

17. TCP vs UDP: what’s the difference and which protocol is better? TCP vs UDP: What’s the Difference and Which Protocol Is Better? URL: https://www.avast.com/c-tcp-vs-udp-difference (дата звернення: 17.05.2024).

18. time – Time access and conversions. Python documentation. URL: https://docs.python.org/uk/3/library/time.html (дата звернення: 27.05.2024).

19. User datagram protocol (UDP) - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/user-datagram-protocol-udp/ (дата звернення: 17.05.2024).

20. Volume 1: Overview / S. Bishop et al. TCP, UDP, and Sockets: rigorous and experimentally-validated behavioural specification. Canberra, 2005.

21. Welcome to python.org. Python.org. URL: https://www.python.org/ (дата звернення: 17.05.2024).

22. What is transmission control protocol (TCP)? - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/ (дата звернення: 17.05.2024).

ДОДАТОК А

ТЕКСТ ПРОГРАМНОГО МОДУЛЯ

«РЕАЛІЗАЦІЯ ЛОГІКИ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМИ»

УКР.НТУУ“КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-02

Аркушів 6

Київ – 2024

```
import socket
import select
import time
from itertools import chain
from operator import sub

ip_address = "127.0.0.1"
port = 50001
timeout_duration = 5
buffer_size = 50000
win_size = 100
connected_clients = []

def process_final_window():
    global output_file, require_reorder, packet_indices, packet_data, client_addr, ack_buffer, seq_number

    print("Data payload received!")
    print("Received packet indices: " + str(packet_indices))

    if require_reorder:
        temp_data = packet_data[:]
        temp_indices = packet_indices[:]
        for i in range(len(packet_indices)):
            mod = divmod(temp_indices[i], win_size)
            temp_indices[i] = mod[1]

        for i in range(len(packet_indices)):
            packet_data[i] = temp_data[temp_indices.index(i)]
```

```
            packet_indices.sort()

            print("Reorder needed: True, sorted indices: " + str(packet_indices))
            require_reorder = False
        else:
            print("Reorder needed: Not required")

        for data in packet_data:
            output_file.write(data)

        seq_number = 0
        packet_data = []
        packet_indices = []

        ack_buffer.append(b"ACK_ALL")

udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_socket.bind((ip_address, port))

print(f"Server is running on {ip_address}:{port}")

while True:
    print("Awaiting connection...")

    msg, client_addr = udp_socket.recvfrom(buffer_size)
    print(f"Message received from {client_addr[0]}:{client_addr[1]}: {msg.decode()}")

    if msg == b"CONN_REQ":
        connected_clients.append(client_addr)
        udp_socket.sendto(b"ACK", client_addr)
```

```
    print(f"Connection established with {client_addr[0]}:{client_addr[1]}")
elif msg.startswith(b"REQ_TO_SEND"):
    file_name = msg[len(b"REQ_TO_SEND"):].decode()
    print("Verifying client...")
    if client_addr in connected_clients:
        udp_socket.sendto(b"ACK", client_addr)
        print("Client verified. Preparing to receive file...")

        client_win_size = udp_socket.recvfrom(buffer_size)[0]
        if int(client_win_size) != win_size:
            udp_socket.sendto(b"NACK", client_addr)
            print("File transfer failed! Window sizes do not match.")
            break
        else:
            udp_socket.sendto(b"ACK", client_addr)
            print("Window sizes match. Starting file transfer...")

        udp_socket.settimeout(timeout_duration)

        recv_file_name = "recv_" + file_name
        output_file = open(recv_file_name, 'wb')
        seq_number = 0
        packet_data = []
        packet_indices = []
        ack_buffer = []
        final_iteration = False
        require_reorder = False

        start_time = time.time()
```

```
        while True or final_iteration:
            if final_iteration:
                process_final_window()
                final_iteration = False
                print("\nSaving file. Please wait...")

            else:
                ready_sockets = select.select([udp_socket], [], [], timeout_duration)
                if ready_sockets[0]:
                    try:
                        data, sender_addr = udp_socket.recvfrom(buffer_size)

                        if data == b"REQ_ACK":
                            if seq_number != win_size and seq_number != 0:
                                missing_packets = list(chain.from_iterable(
                                    (packet_indices[i] + d for d in range(1, diff)) for i, diff in
                                    enumerate(map(sub, packet_indices[1:], packet_indices)) if diff
> 0))

                                i = len(missing_packets)
                                while i:
                                    i -= 1
                                    udp_socket.sendto(b"NACK"                                    +
str(missing_packets[i]).encode(), client_addr)
                                    print(f"Packet     #{missing_packets[i]}     lost.     Requesting
resend...")
                                    data, sender_addr = udp_socket.recvfrom(buffer_size)

                                    packet_data.insert(int(missing_packets[i]), data)
                                    seq_number += 1
```

```
                if i < packet_indices[-1]:
                    require_reorder = True
                packet_indices.append(missing_packets[i])

            process_final_window()

        else:
            udp_socket.sendto(ack_buffer[-1], client_addr)
    else:
        packet_data.append(data)
        seq_number += 1

        data, sender_addr = udp_socket.recvfrom(buffer_size)

        rec_packet_num = int(data[1:])

        if packet_indices:
            if rec_packet_num < packet_indices[-1]:
                require_reorder = True

        packet_indices.append(rec_packet_num)

        if data[0] == 70:
            final_iteration = True

        if seq_number >= win_size:
            process_final_window()
except Exception as e:
    print("Timeout reached. File transfer failed.")
    output_file.close()
```

```
                print("Closing file...")
                break
            else:
                end_time = time.time()
                print(f"File {recv_file_name} saved successfully!")
                output_file.close()
                print("Closing file...")
                transfer_duration = end_time - start_time
                print(f"Time taken to transfer file: {transfer_duration:.2f} seconds")
                break

        break
    else:
        udp_socket.sendto(b"NACK", client_addr)
        print("Verification failed! Client must establish connection with server first.")

print("Shutting down server...")
udp_socket.close()
```