Міністерство освіти і науки України  
Національний технічний університет України

«Київський політехнічний інститут ім. І. Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Лабораторна робота №2

з дисципліни «Інформаційне забезпечення безпеки комп’ютерних систем»

Тема: «Порівняння протоколів TCP та UDP у клієнт-серверній моделі на Python»

Виконав:

студент 4-го курсу, НН ІАТЕ

групи ТР-23

Ровний Г. О.

Перевірив:

Тітов В.М.

КИЇВ 2025

**Мета роботи:**

– відпрацювати навички створення мережевих додатків на основі протоколу UDP, визначити переваги та недоліки UDP і TCP та провести порівняння швидкості, надійності та поведінки при втраті пакетів:

— реалізувати простий сервер UDP, який приймає запити від клієнта;

— протестувати роботу сервера та клієнта;

— здійснити замір часу передачі для обох протоколів.

**Поставлене завдання:**

Реалізація UDP-сервера

1) Створіть файл server\_udp.py.

2) Імпортуйте модуль і створіть сокет.

3) Запустіть сервер: server.py.

**2.** Аналогічно як в лабораторній роботі №1 - реалізуйте UDP-клієнта (створіть файл client\_udp.py.)

**3.** Оцініть поведінку при підключенні кількох клієнтів підряд.

**4.** Оцінити час передачі для обох протоколів (за допомогою time.time()).

**5.** Зробіть імітацію втрати пакету в UDP (сервер не відповідає).

**6.** Змініть повідомлення, додайте цикл з 100 запитів.

**7.** Зробіть логування у файл: час, тип протоколу, статус відповіді.

**Результат виконання роботи**

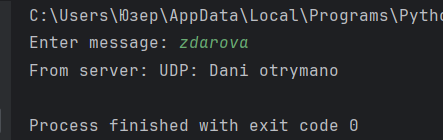
Створюємо файл server\_udp.py. Імпортуємо модуль і створюємо сокет. Після запускаємо сервер: server\_udp.py:

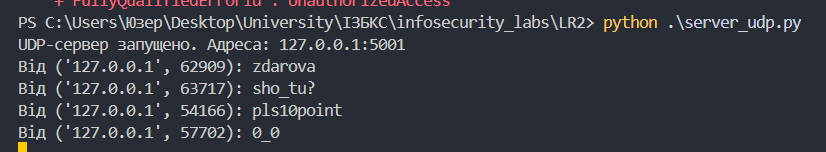
import socket  
  
server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
server.bind(('127.0.0.1', 5001))  
  
print("UDP-сервер запущено. Адреса: 127.0.0.1:5001")  
  
while True:  
 data, addr = server.recvfrom(1024)  
 print(f"Від {addr}: {data.decode()}")  
 server.sendto(b"UDP: Dani otrymano", addr)

Реалізуємо UDP-клієнта. Створюємо файл client\_udp.py, та беремо код з минулої лабораторної роботи:

import socket  
  
HOST = '127.0.0.1'  
PORT = 5001  
  
client = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
message = input("Enter message: ")  
client.sendto(message.encode(), (HOST, PORT))  
  
data, \_ = client.recvfrom(1024)  
print("From server:", data.decode())

Оцінимо поведінку при підключенні кількох клієнтів підряд. Вмикаємо сервер та починаємо відправляти запити від клієнтів. Робимо декілька запитів:



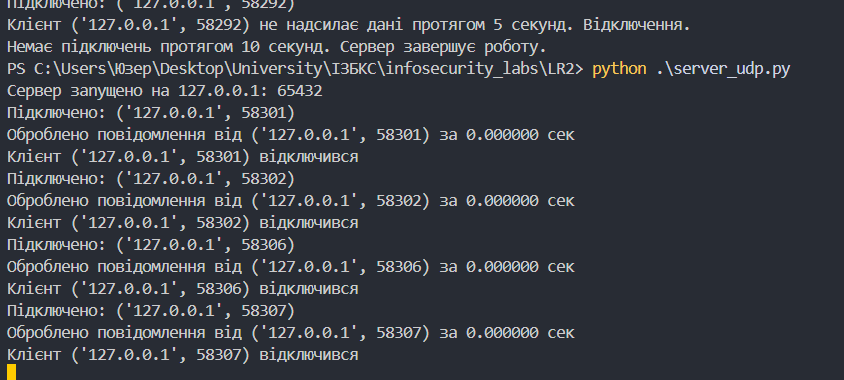


Протокол **UDP** працює без встановлення з'єднання. Сервер просто "слухає" певний порт, готовий приймати дані від будь-кого. Кожен пакет даних, що надходить, є незалежним і містить адресу відправника, що дозволяє серверу надіслати відповідь саме йому. Навіть якщо повідомлення від різних клієнтів приходять одночасно, сервер обробляє їх послідовно. Однак UDP не гарантує, що пакети, відправлені в один і той же час, прийдуть у правильному порядку.

Додамо оцінку часу передачі для обох протоколів. Зробимо імітацію втрати пакетів в UDP. Потім змінимо повідомлення і додамо цикл зі 100 запитів. Насамкінець, зробимо логування у файл:

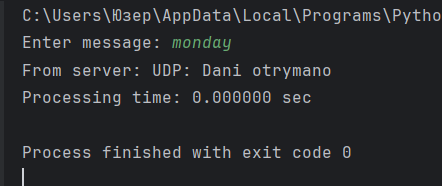
import socket  
import time  
  
HOST = '127.0.0.1'  
PORT = 65432  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  
 s.connect((HOST, PORT))  
 message = input("Enter message: ")  
 start\_time = time.time()  
 s.sendall(message.encode())  
 data = s.recv(1024)  
 end\_time = time.time()  
 delay = end\_time - start\_time  
 print(f"Answer: {data.decode()} | Time={delay:.6f} sec")

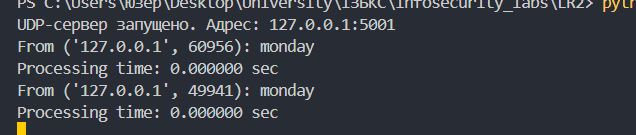
import socket  
import time  
  
HOST = '127.0.0.1'  
PORT = 65432  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  
 s.bind((HOST, PORT))  
 s.listen()  
 s.settimeout(10)  
 print(f"Сервер зanyщено нa {HOST}: {PORT}")  
   
 try:  
 while True:  
 try:  
 conn, addr = s.accept()  
 except socket.timeout:  
 print("Немає підключень протягом 10 секунд. Сервер завершує роботу.")  
 break  
 except Exception as e:  
 print(f"Помилка при прийомі підключення: {e}")  
 continue  
 with conn:  
 print(f"Підключено: {addr}")  
 conn.settimeout(5)  
 while True:  
 try:  
 data = conn.recv(1024)  
 start\_time = time.time()  
 except socket.timeout:  
 print(f"Клієнт {addr} не надсилає дані протягом 5 секунд. Відключення.")  
 break  
 except Exception as e:  
 print(f"Помилка при отриманні даних від {addr}: {e}")  
 break  
  
 if not data:  
 print(f"Клієнт {addr} відключився")  
 break  
  
 response = data.upper()  
 try:  
 conn.sendall(response)  
 end\_time = time.time()  
 delay = end\_time - start\_time  
 print(f"Оброблено повідомлення від {addr} за {delay:.6f} сек")  
 except Exception as e:  
 print(f"Помилка при відправці даних {addr}: {e}")  
 break  
 except KeyboardInterrupt:  
 print("\nСервер зупинено користувачем")



Також оновимо попередній код наступним чином:

import socket  
import time  
  
server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
server.bind(('127.0.0.1', 5001))  
  
print("UDP-сервер запущено. Адрес: 127.0.0.1:5001")  
  
while True:  
 data, addr = server.recvfrom(1024)  
 start\_time = time.time()  
  
 message = data.decode()  
 print(f"From {addr}: {message}")  
  
 server.sendto(b"UDP: Dani otrymano", addr)  
  
 end\_time = time.time()  
 delay = end\_time - start\_time  
 print(f"Processing time: {delay:.6f} sec")



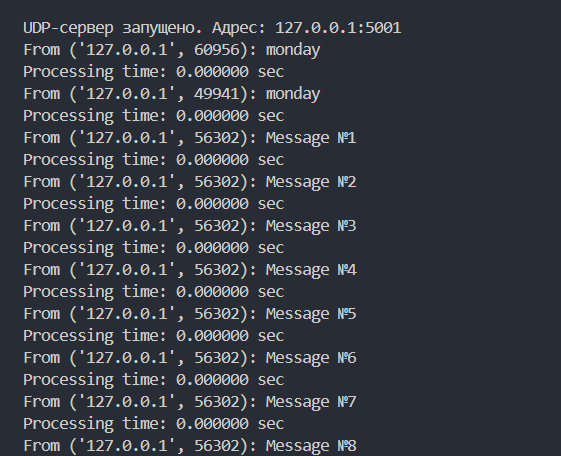


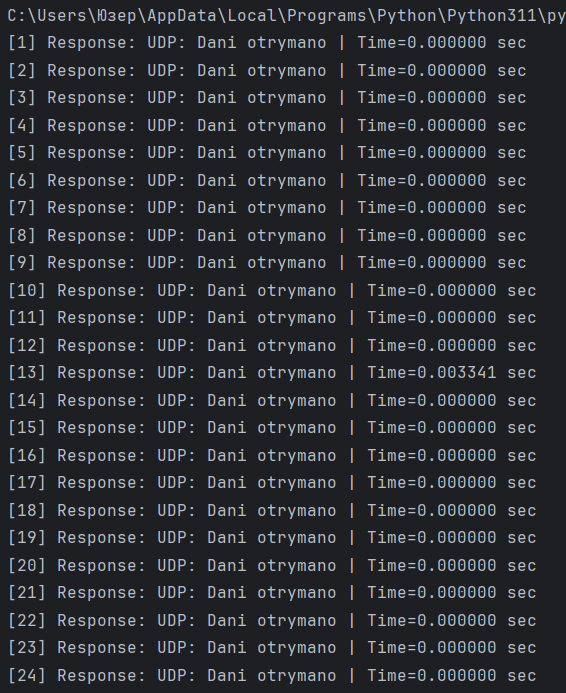
У результаті виконання помітно дуже малий час передачі, але у UDP час передачі має бути менше. Це можна побачити, якщо збільшимо кількість цифр після коми.

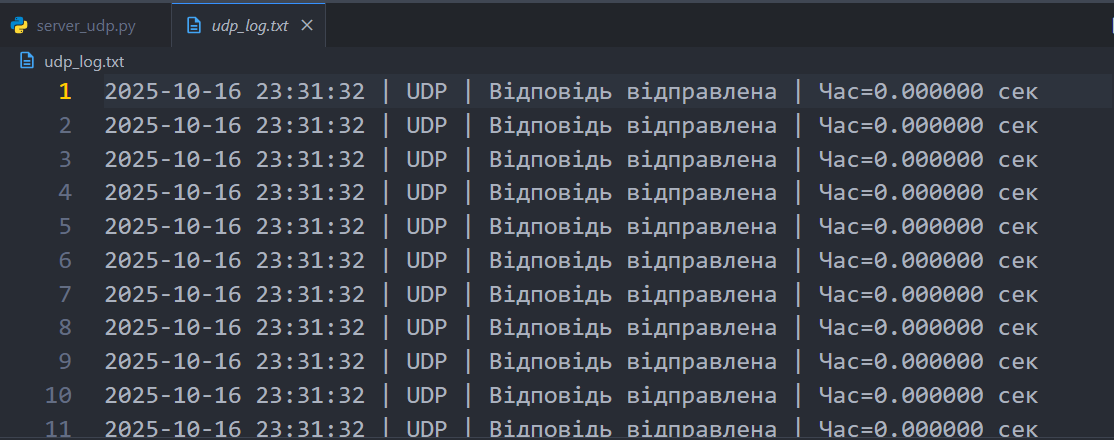
Додамо тепер імітацію втрати пакетів, цикл з 100 повідомлень та логування(udp\_log.txt):

import socket  
import time  
import random  
  
HOST = "127.0.0.1"  
PORT = 5001  
LOG\_FILE = "udp\_log.txt"  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as server:  
 server.bind((HOST, PORT))  
 server.settimeout(10)  
 print(f"UDP-сервер запущено на {HOST}:{PORT}")  
  
 while True:  
 try:  
 data, addr = server.recvfrom(1024)  
 except socket.timeout:  
 print("Немає сигналів більше 10 секунд. Сервер завершує роботу.")  
 break  
  
 start\_time = time.time()  
 message = data.decode()  
 print(f"Отримано від {addr}: {message}")  
  
 if random.random() < 0.1:  
 print("Імітація випадкової втрати пакету, відповіді не буде")  
 status = "Випадкова втрата пакету"  
 else:  
 reply = "UDP: Дані отримано"  
 server.sendto(reply.encode(), addr)  
 status = "Відповідь відправлена"  
  
 end\_time = time.time()  
 delay = end\_time - start\_time  
 print(f"Час обробки: {delay:.6f} сек")  
  
 with open(LOG\_FILE, "a", encoding="utf-8") as log:  
 log.write(f"{time.strftime('%Y-%m-%d %H:%M:%S')} | UDP | {status} | Час={delay:.6f} сек\n")

import socket  
import time  
  
HOST = "127.0.0.1"  
PORT = 5001  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as client:  
 for i in range(1, 101):  
 message = f"Message №{i}"  
 start\_time = time.time()  
  
 client.sendto(message.encode(), (HOST, PORT))  
  
 client.settimeout(2)  
 try:  
 data, addr = client.recvfrom(1024)  
 status = data.decode()  
 except socket.timeout:  
 status = "No response. Packets lost."  
  
 end\_time = time.time()  
 print(f"[{i}] Response: {status} | Time={end\_time - start\_time:.6f} sec")







Таким чином, отримали UDP сервер, який обслуговує 100 запитів, імітує втрату пакетів, вимірює час передачі та вимикається після 10 секунд бездіяльності.

**Відповіді на запитання**

1. Які основні відмінності між TCP та UDP за принципом роботи?

TCP є протоколом, орієнтованим на встановлення з'єднання. Перед початком обміну даними він ініціює сесію між клієнтом і сервером. Передача інформації відбувається у вигляді впорядкованого потоку байтів. Протокол гарантує доставку даних у коректній послідовності, забезпечує контроль цілісності за допомогою механізмів перевірки помилок та ініціює повторну передачу втрачених сегментів.

UDP, натомість, функціонує без встановлення з'єднання. Дані інкапсулюються в окремі, незалежні пакети (датаграми) та надсилаються без попереднього узгодження сесії. UDP не надає гарантій доставки, не контролює послідовність отримання пакетів та не має вбудованих механізмів виправлення помилок. Це забезпечує вищу швидкість передачі, але знижує надійність порівняно з TCP.

2. Чи можна отримати підтвердження доставки в UDP? Як це реалізувати?

Стандартна специфікація протоколу UDP не містить вбудованого механізму підтвердження доставки пакетів. Однак такий функціонал може бути реалізований на прикладному рівні (application layer). Загальна схема реалізації передбачає, що після надсилання пакета даних клієнт очікує на отримання відповідного пакета-підтвердження від сервера. Якщо таке підтвердження не надходить протягом визначеного часового інтервалу (тайм-ауту), клієнтська програма може ініціювати повторну відправку пакета.

3. Що трапиться, якщо сервер не відповість на UDP-запит?

У випадку, коли сервер не надсилає відповідь на UDP-запит, клієнт не отримує жодного автоматичного сповіщення про помилку на рівні транспортного протоколу. Відсутність відповіді може бути спричинена втратою запиту на шляху до сервера, втратою відповіді на зворотному шляху, або тим, що сервер не обробив запит. Відповідальність за виявлення та обробку таких ситуацій (наприклад, шляхом повторення запиту) повністю покладається на логіку, реалізовану в додатку.

4. Чи можливо реалізувати власний механізм підтвердження доставки в UDP?

Так, можливо. Для цього реалізують логіку на стороні клієнта і сервера. Кожен пакет має унікальний ідентифікатор, сервер надсилає підтвердження отримання конкретного пакета, клієнт контролює таймаут і повторно відправляє пакет у разі відсутності підтвердження. Цей підхід дозволяє зробити передачу через UDP більш надійною. В моєму останньому коді наведено приклад власного механізму підтвердження доставки, але без повторного надсилання пакета. Щоб код виконував повторне відсилання після імітації випадкової втрати пакетів, треба додати до нього повторне відправлення повідомлення **.**

**Висновок:**

У результаті виконання лабораторної роботи було набуто навичок створення мережевих додатків на основі протоколу UDP а також переваги та недоліки UDP і TCP та провів порівняння швидкості, надійності та поведінки при втраті пакетів. Реалызовано простий сервер UDP, який приймає запити від клієнта та протестовано роботу сервера та клієнта і зроблено замір часу передачі для обох протоколів.