**Bansilal Ramnath Agarwal Charitable Trust’s**



# Vishwakarma Institute of Technology,Pune-37

*(An Autonomous Institute of Savitribai Phule Pune University)*

**Department of Artificial Intelligence and Data Science**

|  |  |
| --- | --- |
| **Division** | A |
| **Batch** | 3 |
| **GR-no** | 12311951 |
| **Roll no:** | 25 |
| **Name** | Archit Bagad |

**Description :**

1. Sender Module:

● Establishes a UDP socket for communication.

● Reads the specified file in chunks.

● Sends each chunk as a UDP packet to the receiver.

● Handles potential packet loss or out-of-order delivery.

2. Receiver Module:

● Creates a UDP socket and binds it to a specified port.

● Listens for incoming UDP packets containing file data.

● Reassembles the received packets and saves the file locally.

● Implements error checking and packet reordering to ensure

reliability.

3. UDP (User Datagram Protocol):

● UDP is a connectionless transport protocol that operates over IP.

● Unlike TCP, UDP does not establish a connection before data

transfer.

● It is suitable for scenarios where low latency and simplicity are more

important than reliability.

4. File Transfer Using UDP:

● In a UDP-based file transfer, the client and server exchange

datagrams (packets) without establishing a connection.

● The server waits for incoming datagrams, and the client sends the

file data in chunks.

5. Steps for UDP File Transfer:

● Server Side:

1. The server starts and waits for a filename from the client.

2. Upon receiving the filename, the server checks if the file exists.

3. If the file exists, the server reads the file and sends it to the client in

encrypted buffers until the end of the file is reached.

4. The end of the file is marked by an EOF (End of File) character.

5. The server decrypts the received buffers.

6. If the file does not exist, the server sends a “File Not Found” message.

● Client Side:

7. The client sends the filename to the server.

8. Upon receiving the file, the client decrypts the data and reconstructs the

original file.

6. Encryption:

● XOR encryption is used in this example for simplicity.

● Each character in the file is XOR-ed with a secret key (e.g., ‘S’)

before transmission.

● The same key is used for decryption on the receiving side.

7. Packet Size:

● The maximum size of a UDP packet is 64 kB.

● For larger files, consider breaking them into smaller chunks.

8. Error Handling:

● UDP does not guarantee reliable delivery or order preservation.

● Implement error handling and retransmission logic if needed.

Code:

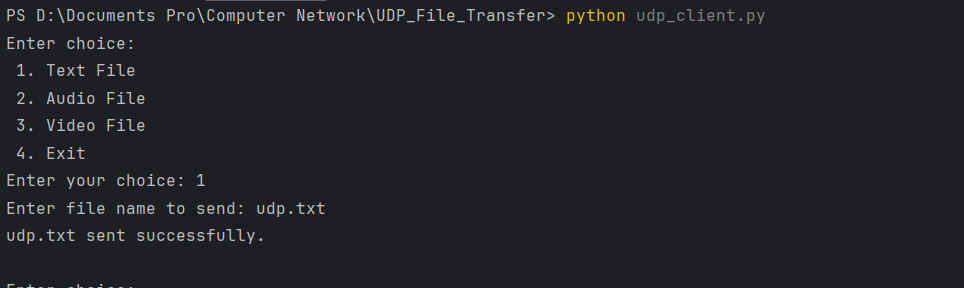
**Server.py :**  
  
import socket  
  
BUFFER\_SIZE = 70000  
PORT = 8000  
  
  
def receive\_file(sock, client\_addr, file\_name, file\_size):  
 *"""Function to receive a file from the client and save it."""* print(f"Receiving {file\_name}...")  
  
 data, \_ = sock.recvfrom(file\_size)  
  
 with open(file\_name, "wb") as f:  
 f.write(data)  
  
 print(f"{file\_name} received successfully.\n")  
  
  
def main():  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 sock.bind(("0.0.0.0", PORT))  
  
 print("Server is running and waiting for connections...")  
  
 while True:  
 choice, client\_addr = sock.recvfrom(1)  
 choice = int.from\_bytes(choice, byteorder='big')  
  
 if choice == 1: # Text file  
 file\_name, \_ = sock.recvfrom(1024)  
 file\_name = file\_name.decode()  
  
 receive\_file(sock, client\_addr, file\_name, BUFFER\_SIZE)  
  
 elif choice == 2: # Audio file  
 file\_name, \_ = sock.recvfrom(1024)  
 file\_name = file\_name.decode()  
  
 receive\_file(sock, client\_addr, file\_name, BUFFER\_SIZE)  
  
 elif choice == 3: # Video file  
 file\_name, \_ = sock.recvfrom(1024)  
 file\_name = file\_name.decode()  
  
 receive\_file(sock, client\_addr, file\_name, BUFFER\_SIZE)  
  
 elif choice == 4: # Exit  
 print("Closing server...")  
 sock.close()  
 break  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

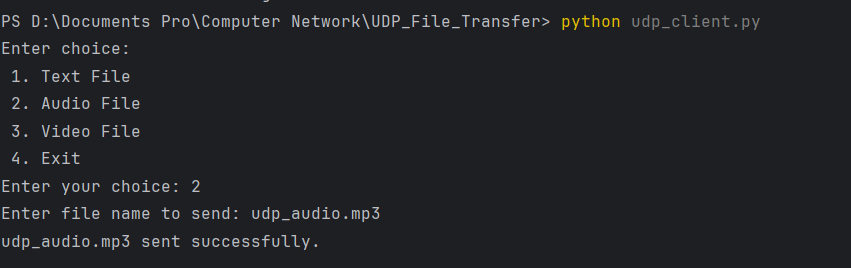
**Client.py:**

import socket  
  
SERVER\_IP = "127.0.0.1" # Change to the server's IP if needed  
PORT = 8000  
BUFFER\_SIZE = 70000  
  
def send\_file(sock, file\_name, server\_address):  
 *"""Function to send a file to the server."""* try:  
 with open(file\_name, "rb") as f:  
 file\_data = f.read()  
  
 sock.sendto(file\_name.encode(), server\_address)  
 sock.sendto(file\_data, server\_address)  
  
 print(f"{file\_name} sent successfully.\n")  
  
 except FileNotFoundError:  
 print("File not found. Please check the file name.")  
  
  
def main():  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_address = (SERVER\_IP, PORT)  
  
 while True:  
 print("Enter choice:\n 1. Text File\n 2. Audio File\n 3. Video File\n 4. Exit")  
 choice = int(input("Enter your choice: "))  
  
 sock.sendto(choice.to\_bytes(1, byteorder='big'), server\_address)  
  
 if choice in [1, 2, 3]:  
 file\_name = input("Enter file name to send: ")  
 send\_file(sock, file\_name, server\_address)  
  
 elif choice == 4:  
 print("Closing client...")  
 sock.close()  
 break  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Result :

Client\_side :





Server\_side :

