**UDP File Transfer Documentation**

**1. Overview**

This project implements a simple file transfer system using the **User Datagram Protocol (UDP)**. Unlike TCP, UDP is **connectionless and unreliable**, meaning that packet delivery, order, and integrity are not guaranteed. Therefore, additional logic is added to handle potential **errors**, such as packet loss, corruption, and out-of-order delivery.

**2. Components**

The system has two main components:

* **Sender (Client)**: Reads and sends the file in chunks over UDP.
* **Receiver (Server)**: Listens for incoming packets and reconstructs the file.

**3. Workflow**

**Sender Side:**

### Step-by-Step Process:

1. **Open the File**:
   * Opens the file in binary mode (rb) to read its contents.
2. **Split the File**:
   * Reads the file in fixed-size chunks (e.g., 1024 bytes).
3. **Add Packet Header**:
   * For each chunk, the sender adds a header that includes:
     + **Sequence number** (to track order)
     + **EOF flag** (to indicate the last packet)
     + **Chunk size**
4. **Send the Packet**:
   * Sends each packet over UDP to the receiver's IP and port.
5. **(Optional) Wait for ACK**:
   * If reliability is added, the sender waits for an acknowledgment (ACK) for each packet.
   * If no ACK is received within a **timeout**, it **retransmits** the packet.
6. **Repeat**:
   * Continues until all chunks are sent and acknowledged.
7. **Close Socket**:
   * After sending all packets (and receiving final ACK), the sender closes the socket.

import socket

import os

import time

CHUNK\_SIZE = 1024

ACK\_TIMEOUT = 2 # seconds

MAX\_RETRIES = 5

def send\_file(filename, receiver\_ip, receiver\_port):

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.settimeout(ACK\_TIMEOUT)

file\_size = os.path.getsize(filename)

total\_chunks = (file\_size + CHUNK\_SIZE - 1) // CHUNK\_SIZE

with open(filename, "rb") as f:

for seq in range(total\_chunks):

chunk = f.read(CHUNK\_SIZE)

eof\_flag = 1 if seq == total\_chunks - 1 else 0

header = f"{seq}|{eof\_flag}|".encode()

packet = header + chunk

retries = 0

while retries < MAX\_RETRIES:

sock.sendto(packet, (receiver\_ip, receiver\_port))

try:

ack\_data, \_ = sock.recvfrom(1024)

ack\_seq = int(ack\_data.decode())

if ack\_seq == seq:

print(f"ACK received for packet {seq}")

break

except socket.timeout:

print(f"Timeout for packet {seq}, retrying...")

retries += 1

else:

print(f"Failed to send packet {seq} after retries. Aborting.")

sock.close()

return

print("File sent successfully.")

sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) != 4:

print("Usage: python sender.py <receiver\_ip> <receiver\_port> <filename>")

else:

send\_file(sys.argv[3], sys.argv[1], int(sys.argv[2]))

**Receiver Side:**

### Step-by-Step Process:

1. **Bind to UDP Port**:
   * Opens a UDP socket and listens on a specific port for incoming packets.
2. **Wait for Packets**:
   * Receives UDP packets as they arrive.
3. **Read Header**:
   * Extracts:
     + **Sequence number**
     + **EOF flag**
     + **Chunk size**
4. **Store the Data**:
   * Keeps received data in a dictionary or buffer, using the sequence number as the key.
5. **Send ACK (Optional)**:
   * Sends an ACK back to the sender for each packet received.
6. **Check for Completion**:
   * If the EOF flag is detected and all previous sequence numbers are received:
     + Assembles the file in correct order.
     + Writes the file to disk.
7. **Handle Errors**:
   * Ignores **duplicate packets** (already received SEQ\_NO).
   * **Reorders out-of-order** packets using SEQ\_NO.
   * Waits or requests **retransmission** if a packet is missing (if ACK system is used).
8. **Close and Save**:
   * Closes the socket and writes the final file to disk

import socket

def receive\_file(listen\_port, output\_filename):

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(('', listen\_port))

print(f"Listening on port {listen\_port}...")

received\_data = {}

expected\_packets = None

while True:

packet, addr = sock.recvfrom(2048)

try:

header\_end = packet.index(b'|', packet.index(b'|') + 1)

header = packet[:header\_end].decode()

seq\_str, eof\_str = header.split('|')

seq = int(seq\_str)

eof = int(eof\_str)

data = packet[header\_end + 1:]

if seq not in received\_data:

received\_data[seq] = data

print(f"Received packet {seq}")

# Send ACK

ack = f"{seq}".encode()

sock.sendto(ack, addr)

if eof == 1:

print("Last packet received. Writing file...")

break

except Exception as e:

print("Error processing packet:", e)

# Write the file in correct order

with open(output\_filename, "wb") as f:

for i in range(len(received\_data)):

f.write(received\_data[i])

print("File received and saved successfully.")

sock.close()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) != 3:

print("Usage: python receiver.py <listen\_port> <output\_filename>")

else:

receive\_file(int(sys.argv[1]), sys.argv[2])

**4. Packet Structure**

[SEQ\_NO][EOF][CHUNK\_SIZE][DATA]

- SEQ\_NO: Sequence number (integer)

- EOF: 1 if it's the last packet, 0 otherwise

- CHUNK\_SIZE: Number of bytes in the data

- DATA: Actual bytes from the file

**5. Error Handling**

UDP does not guarantee delivery or order, so we handle errors manually.

**Implemented Error Handling Techniques:**

| **Error Type** | **Description** | **Handling Method** |
| --- | --- | --- |
| **Packet Loss** | Some packets may never arrive | Use timeouts + ACKs and retransmissions |
| **Out-of-Order Packets** | Packets may arrive in any order | Use sequence numbers to reorder |
| **Duplicate Packets** | Packets may be resent accidentally | Track received SEQ\_NO to ignore duplicates |
| **Corruption** | Data could be corrupted in transit | Add checksum/CRC (optional) and discard bad packets |
| **Missing Final Packet** | Last packet might be lost | Receiver expects EOF flag, re-requests if not received |

**Optional Enhancements:**

* **Checksum/CRC** for data integrity.
* **Sliding Window Protocol** for better performance and flow control.
* **Selective Repeat ARQ** for efficient retransmission.

**6. Configuration Parameters**

* CHUNK\_SIZE: Size of each packet’s data payload (e.g., 1024 bytes).
* TIMEOUT: Duration (in seconds) before resending a packet if no ACK is received.
* MAX\_RETRIES: Number of retries before aborting transmission.
* PORT: UDP port number used for communication.
* ACK\_REQUIRED: Boolean flag to enable/disable ACK-based reliability.

**7. Usage**

**Sender**

python sender.py <receiver\_ip> <receiver\_port> <filename>

**Receiver**

python receiver.py <listen\_port>

**8. Limitations**

* UDP is not suitable for very large files or unreliable networks unless enhanced.
* The current implementation may not scale well without threading or async I/O.
* No encryption or authentication is provided.

**9. Future Improvements**

* Use **reliable UDP** techniques (like RUDP).
* Add **GUI interface** using Tkinter or PyQt.
* Implement **multi-threading** to handle concurrent transfers.
* Support **resume interrupted transfers**.