1. Introduction & Author

This project implements a basic UDP client-server file transfer system with error handling using Reliable Data Transfer (RDT) 2.2. It is built over an unreliable UDP channel that can introduce bit errors in packets and ACKs.

Written by: Joseph Nguyen 2/21/2025

2. Phase 2

This project implements **Reliable Data Transfer (RDT) 2.2** using **UDP** in an unreliable network environment that introduces **bit errors in packets and ACKs**. The implementation includes:

- **Checksum-based error detection**
- **ACK validation and retransmissions**
- **Handling ACK corruption**
- **Timeout-based packet retransmission**
- **Performance measurement of transfer times under different error rates**

3. Phase 2 (Extra Credit)

The extra credit phase enhances the Reliable Data Transfer (RDT 2.2) protocol by incorporating advanced networking features, improved error detection, and a high-quality GUI for real-time monitoring. A graphical interface was implemented to visualize the file transfer process, including FSM (Finite State Machine) state transitions for both sender and receiver, live progress tracking, and real-time image preview as packets arrive. To simulate real-world network conditions, random transmission delays (0-500ms) and an adaptive timeout mechanism were introduced to dynamically adjust retransmission timing based on observed delays. Additionally, CRC-16 was implemented as an alternative to XOR checksum, improving error detection and reducing packet corruption. A multi-threaded approach was used to optimize data transmission, ensuring concurrent packet sending and acknowledgment processing, leading to reduced latency and improved performance. Finally, a comparative analysis was conducted to evaluate the impact of these enhancements on transmission time, error rates, and retransmissions under varying loss conditions (0% to 60%).

4. System Overview

4.1 Components

The system consists of two main components:

1. **Client (Sender)**
    - Reads and splits the **JPEG image file** into **1024-byte packets**.
    - Appends a **4-byte sequence number** and **1-byte checksum** to each packet.
    - Sends packets via UDP **one at a time**, waiting for an ACK before sending the next packet.
    - Retransmits packets if **ACKs are missing or corrupted**.
2. **Server (Receiver)**
    - Receives and validates packets using **checksum verification**.
    - Sends **ACKs** for valid packets.
    - Detects **corrupt or duplicate packets** and requests retransmissions.
    - Reassembles and saves the received **JPEG image** file.

5. Implementation Details

5.1 Reliable Data Transfer (RDT) 2.2

Client-Side Functionality:

1. **Read BMP file and split into packets** (1024 bytes each).
2. **Calculate checksum** for each packet.
3. **Send the packet with a sequence number** over UDP.
4. **Wait for an ACK**:
    - If ACK is correct, send the next packet.
    - If ACK is missing/corrupted, retransmit the packet.
5. **Continue until all packets are sent**.

Server-Side Functionality:

1. **Receive incoming packets and extract data.**
2. **Validate checksum**:
    - If correct, send an **ACK** and store the packet.
    - If incorrect, discard the packet and do not ACK.
3. **Send ACK with sequence number** to acknowledge successful receipt.
4. **Reassemble packets into an output BMP file** once all packets are received.

6. Error Handling

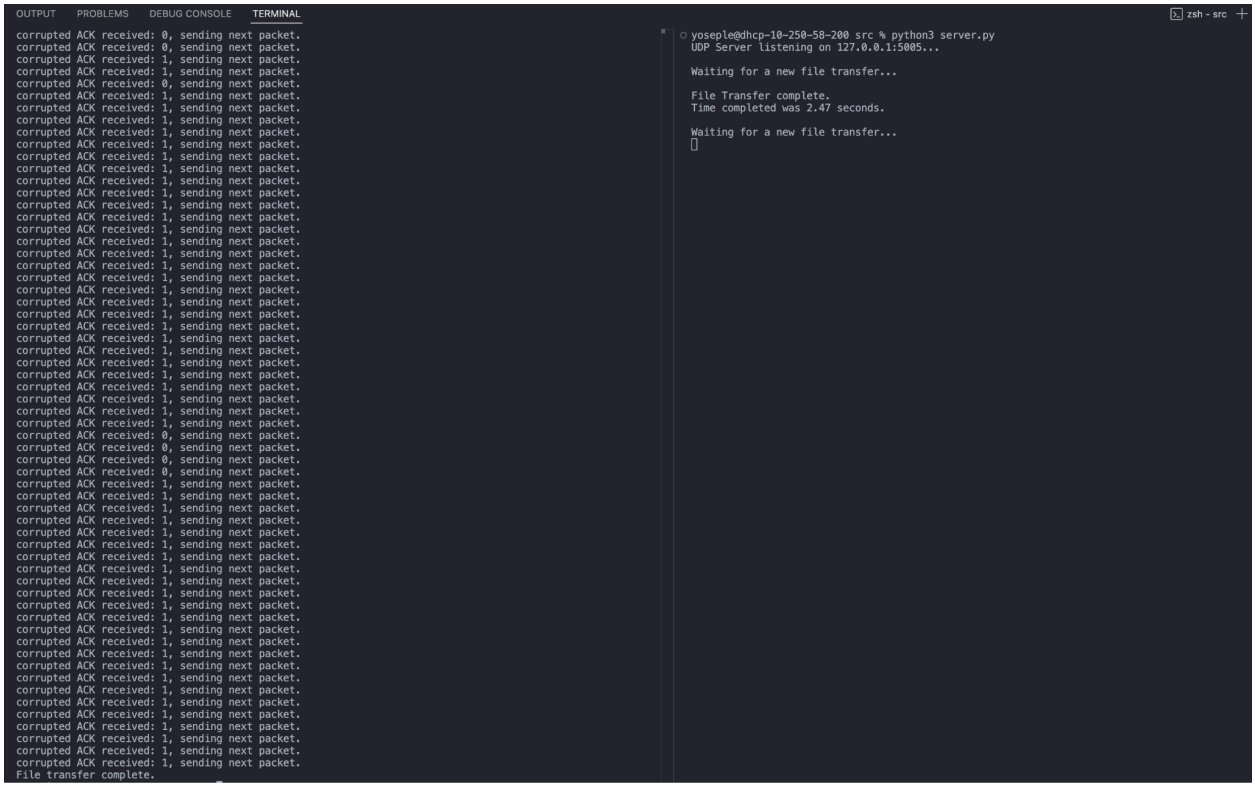
6.1 Checksum-Based Error Detection

- Uses **XOR checksum** to verify packet integrity.
- If checksum validation fails, the packet is **discarded**.

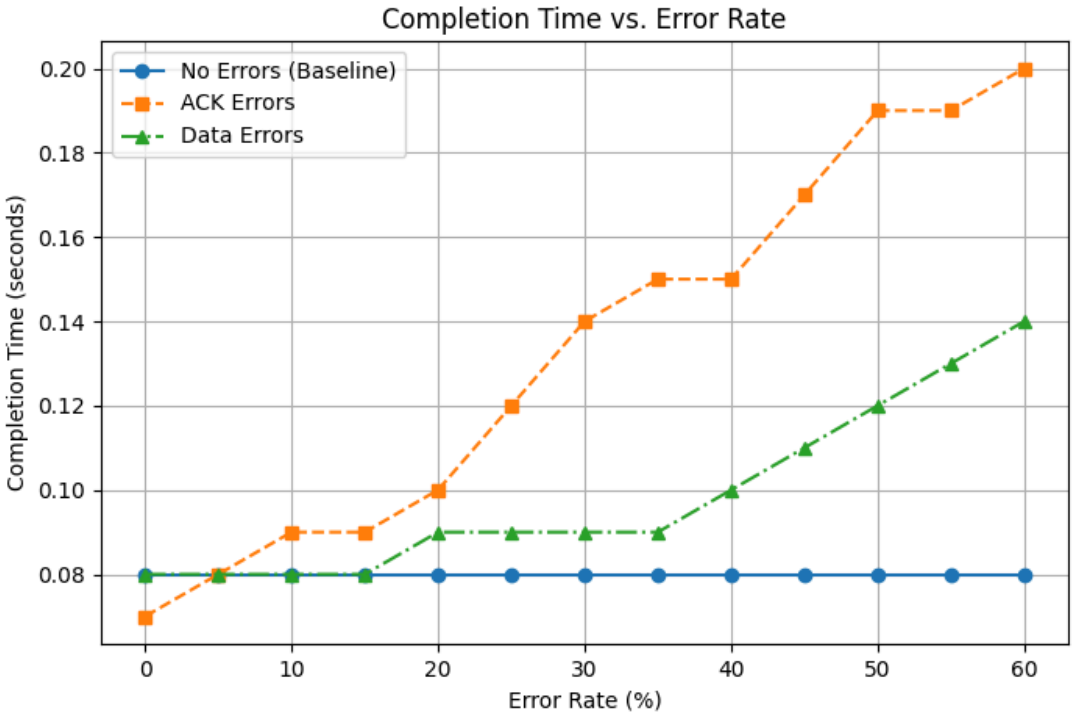
6.2 Handling ACK Corruption

- **Depending on the probability of ACK corruption**.
- If the client receives a **corrupted ACK**, it **retransmits the last packet**.

6.3 Timeout-Based Packet Retransmission

- If timeout occurs, the **last packet is retransmitted**.
- 

7. Performance Evaluation



8. Steps to Execute Program

6.1 Running the Client and Server

1. **Start the server** (First Terminal):
   python3 server.py
2. **Start the client** (Second Terminal):
   python3 client.py
3. The client **transmits the BMP file**, and the **server receives and reassembles it**.
4. The **client simulates packet corruption at increasing error rates**.
5. The **server prints "File Transfer Complete" and logs the completion time**.

6.2 Generating the Performance Graph

1. After running `server.py` and `client.py`,
2. Manually get data by inputting the corruption % in increments of 5
3. Execute: After getting the data from step 2
   python3 plot.py
4. The script generates `performance_plot.png`, showing **how error rates impact completion time**