# TCP Project Report

## 1. Introduction

Socket programming is a way of enabling communication between two nodes on a network. It allows for the creation of client-server applications where data can be sent and received over a network using sockets. A socket is an endpoint for sending or receiving data, and it is bound to a port number so that the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) can identify the application to which the data is destined. In this project, we implemented a TCP-based file transfer application that ensures reliable communication using error detection and retransmission mechanisms.

## 2. Steps for Socket Implementation

### 2.1 Server Implementation

1. 1. Create a TCP socket using the socket library.  
   2. Bind the socket to a specific IP address and port.  
   3. Listen for incoming client connections.  
   4. Accept a client connection and handle file transfer requests.  
   5. Fragment the requested file into segments and send each segment with a checksum.  
   6. Handle retransmission requests for corrupted or lost segments.  
   7. Close the connection after the file transfer is complete.

### 2.2 Client Implementation

1. 1. Create a TCP socket and connect to the server.  
   2. Send a file request to the server.  
   3. Receive file segments from the server and verify their integrity using checksums.  
   4. Request retransmission for corrupted or missing segments.  
   5. Reassemble the file from the received segments and save it locally.  
   6. Close the connection after the file transfer is complete.

## 3. Problems Encountered and Solutions

### 3.1 Problem: Checksum Mismatch

During the file transfer, some segments were corrupted due to simulated errors, resulting in checksum mismatches.

Solution: The client requested retransmission of the corrupted segments, and the server resent them until they were received correctly.

### 3.2 Problem: Connection Timeout

The client experienced connection timeouts when the server took too long to respond.

Solution: Implemented a retry mechanism with exponential backoff to handle connection timeouts.

## 4. Directory Structure

The project directory is organized as follows:  
└── mazens1-tcp/  
 ├── README.md  
 ├── app.py  
 ├── client.py  
 ├── requirements.txt  
 ├── server.py  
 ├── utils.py  
 ├── downloads/  
 │ ├── README.md  
 │ ├── app.py  
 │ └── client.py  
 ├── templates/  
 │ └── index.html  
 └── uploads/  
 ├── README.md  
 ├── app.py  
 └── client.py

### 4.1 File Descriptions

1. 1. README.md: Provides an overview of the project and usage instructions.  
   2. app.py: Implements the Flask web application for managing file transfers.  
   3. client.py: Contains the TCP client implementation for requesting and receiving files.  
   4. requirements.txt: Lists the dependencies required for the project.  
   5. server.py: Contains the TCP server implementation for handling file transfer requests.  
   6. utils.py: Provides utility functions for file fragmentation, checksum calculation, and error simulation.  
   7. downloads/: Directory for storing downloaded files.  
   8. templates/: Contains the HTML templates for the web application.  
   9. uploads/: Directory for storing uploaded files.