Report: Implementation of a Reliable Data Transmission Protocol Using C Sockets

Introduction

This report details the implementation of a reliable data transmission protocol using C sockets. The protocol is designed to handle packet-based communication between a client and a server, ensuring that all packets are correctly received and acknowledged. The implementation includes mechanisms for detecting and handling packet loss or unexpected sequences.

Objectives

The main objectives of this project were:

- To establish a reliable connection between a client and a server using TCP sockets.
- To implement a protocol that ensures packets are received in order and are acknowledged by the receiver.
- To handle packet loss by incorporating a timeout mechanism that allows for retransmission of lost packets.
- To implement an acknowledgment system where the client explicitly acknowledges each received packet.

Implementation Details

Server-Side Implementation

The server is responsible for sending packets to the client and awaiting acknowledgments for each packet before proceeding to send the next one. The key components of the server implementation are:

1. **Socket Creation and Binding**:
    - A TCP socket is created using `socket(AF_INET, SOCK_STREAM, 0)`.
    - The socket is bound to a specific port (8080) using `bind()`.
2. **Listening for Connections**:
    - The server listens for incoming connections with `listen()`.
    - Once a connection is established, it accepts the connection using `accept()`.
3. **Packet Transmission**:
    - The server sends packets containing a sequence number using `send()`.
    - After sending each packet, the server waits for an acknowledgment from the client using the `select()` function, which implements a timeout mechanism. This ensures that if the acknowledgment is not received within a specified time, the packet is retransmitted.
4. **End-of-Transmission Message**:

    - After all packets have been sent and acknowledged, the server sends a special "DONE" message to indicate the end of transmission.

```
int server_fd, new_socket;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
char buffer[PACKET_SIZE] = {0};
fd_set readfds;
struct timeval timeout;
...
```

Client-Side Implementation

The client is responsible for receiving packets from the server, verifying the sequence, and sending acknowledgments. The key components of the client implementation are:

1. **Socket Creation**:
    - A TCP socket is created using `socket(AF_INET, SOCK_STREAM, 0)`.
    - The client attempts to connect to the server at a specified IP address and port.
2. **Packet Reception and Acknowledgment**:
    - The client continuously reads incoming packets using `read()`.
    - It checks if the received packet has the expected sequence number.
    - If the sequence number is correct, the client sends an acknowledgment back to the server.
    - If the sequence number does not match, the client prints an error message and does not send an acknowledgment, causing the server to resend the packet.
3. **End-of-Transmission Handling**:
    - If the client receives the "DONE" message, it closes the connection and terminates the program.

```
int sock = 0;
struct sockaddr_in serv_addr;
char buffer[PACKET_SIZE * 4] = {0};
...
```

Error Handling

The implementation includes error handling for the following scenarios:

- **Socket Creation Failure**: If the socket cannot be created, the program prints an error message and exits.
- **Connection Failure**: If the client fails to connect to the server, an error message is printed.

- **Unexpected Packets**: The client checks if the received packet has the expected sequence number. If not, an error message is displayed.
- **Timeout Handling**: The server uses a timeout mechanism to detect lost packets and resend them.

Conclusion

The project successfully implemented a reliable data transmission protocol using C sockets. The client and server communicate efficiently, ensuring that all packets are received in order and acknowledged. The timeout mechanism effectively handles packet loss, and the acknowledgment system ensures data integrity. This implementation can serve as a foundation for more complex communication protocols in networked applications.