Reliable UDP Using My Transfer Protocol (KTP)
Assignment 4 Documentation

1 Introduction

This report presents the design, implementation, and observations of the Reliable UDP communication system based on My Transfer Protocol (KTP). The protocol ensures reliable data transmission over UDP by implementing sliding window mechanisms, acknowledgments, and retransmission strategies.

2 Contents of the Submitted Folder

The submitted folder contains the following components:

- **Makefile**
- **Header File:** `ksocket.h`
- **C Source Codes:**
  - `ksocket.c`
  - `initksocket.c`
- **User Applications:**
  - `user1.c` (Sender Application)
  - `user2.c` (Receiver Application)
- **Test Files:** `input.txt`

3 Makefile

3.1 Usage

- `make` : Builds the static library, initialization process, and user applications.
- `./initksocket` : Starts the initialization process.
- `./user1` and `./user2` : Initiates sender and receiver applications respectively.

3.2 Targets

- `make all` : Builds all components including:
  - `libksocket.a`: Static library for KTP socket implementation.
  - `initksocket`: Initializes shared memory and management threads.
  - `user1`: User application for sending files.
  - `user2`: User application for receiving files.
- `make apps` : Builds only `user1` and `user2` applications.
- `make clean` : Cleans up all object files, executables, and generated files.

3.3 Command Examples

```
make              # Build all components
./initksocket     # Run the initializer
./user1           # Run the sender app
./user2           # Run the receiver app
make clean        # Remove all build artifacts
```

4 Code Structure and Components

4.1 Header File (ksocket.h)

Defines essential data structures, macros, and function prototypes.

4.2 ksocket.c

Implements the application-level API for user processes.

4.3 initksocket.c

Acts as the initialization process and backbone of the protocol, managing core threads.

5 Data Structures

- **window**: Manages both send and receive windows.
- **SM_entry**: Represents each socket in shared memory.
- **SOCK_INFO**: Used by user processes for socket creation and binding.

6 Functional Workflow

6.1 Sender Side (user1)

1. Creates and binds a socket using `k_socket()` and `k_bind()`.
2. Sends files using `k_sendto()`.
3. Sender thread in `initksocket` transmits packets over UDP.

6.2 Receiver Side (user2)

1. Creates and binds a socket.
2. Receives files using `k_recvfrom()`.
3. Receiver thread acknowledges incoming packets and updates receive windows.

7 Testing and Results

The system was tested using files of varying sizes and packet loss probabilities.

7.1 Packet Loss Simulation Results

Packet Loss Probability	No. of Messages	Transmissions	Ratio
0.05	43	56	1.31
0.10	43	60	1.39
0.15	43	65	1.51
0.20	43	72	1.67
0.25	43	79	1.83
0.30	43	84	1.93
0.35	43	100	2.32
0.40	43	118	2.74
0.45	43	122	2.83
0.50	43	126	2.93

7.2 Observations

- As packet loss probability increases, retransmissions increase proportionally.
- The protocol successfully recovers lost packets, ensuring reliable communication.
- Sliding window and acknowledgment handling optimize throughput under low to moderate loss rates.

8 Conclusion

The **KTP protocol** successfully implements **reliable UDP communication** by integrating **shared memory management, process synchronization, and custom flow control mechanisms**. The design ensures **efficient and robust data transmission**, even under simulated adverse network conditions.