Reliable Data Transfer Protocol Documentation

Overview

This document describes a reliable data transfer protocol implemented in Python. The protocol ensures correct and ordered delivery of packets, handling issues like packet loss, corruption, and reordering using acknowledgments, retransmissions, and timeouts.

Components

The protocol consists of four main components:

1. **PacketClass**: Represents an individual packet.
2. **SenderClass**: Manages sending packets and handling retransmissions.
3. **Receiver**: Handles incoming packets and acknowledges them.
4. **NetworkSimulatorClass**: Simulates network conditions, including packet loss, corruption, and reordering.

PacketClass

- **Fields**:
  - `sequence_number`: Identifies the packet order.
  - `packet_data`: Stores the data being sent.
  - `checksum`: Ensures data integrity using an MD5 hash.
- **Methods**:
  - `calculate_checksum(data)`: Generates a checksum for the packet.
  - `is_corrupted()`: Checks if the packet has been altered during transmission.

SenderClass

- **Fields**:
  - `networkSimulator`: Instance of the network simulator.
  - `max_retries`: Maximum number of retransmission attempts.
  - `timeout`: Time to wait before retransmitting a packet.
  - `sequence_number`: Tracks the next packet to send.
  - `sent_packets`: Stores packets awaiting acknowledgment.

    - `ack_queue`: Queue for received acknowledgments.
- **Methods**:
    - `send_packet(data_to_send)`: Creates and sends a packet.
    - `receive_ack(ack_seq)`: Processes received acknowledgments.
    - `handle_acknowledgement()`: Listens for ACKs and triggers retransmissions.
    - `timeout_retransmit()`: Retransmits unacknowledged packets after timeout.
    - `start_sender_thread()`: Runs the acknowledgment handler in a separate thread.

Receiver

- **Fields**:
    - `expected_seq_num`: Tracks the next expected sequence number.
- **Methods**:
    - `receive_packet(packet)`: Processes received packets, detecting corruption and ordering issues.

NetworkSimulatorClass

- **Fields**:
    - `loss_rate`: Probability of packet loss.
    - `corruption_rate`: Probability of packet corruption.
    - `reorder_rate`: Probability of packet reordering.
    - `receiver`: Instance of the receiver class.
    - `queue`: Buffer for processing packets.
- **Methods**:
    - `send_to_network(packet)`: Simulates network behavior.
    - `process_packet(packet)`: Passes packets to the receiver.
    - `start()`: Runs the network simulator in a loop.

Execution Flow

1. The **sender** starts a thread to listen for acknowledgments.
2. The **sender** sends packets with sequence numbers.
3. The **network simulator** applies possible losses, corruption, or reordering.
4. The **receiver** processes packets and sends ACKs for correctly received packets.
5. The **sender** retransmits unacknowledged packets if needed.
6. The process continues until all packets are successfully delivered.

Running the Program

Execute the script to simulate the transmission of 10 packets:

python reliable_transfer.py

Summary

This protocol ensures reliable data transfer using sequence numbers, checksums, and retransmissions. The sender and receiver interact via an unreliable network, using acknowledgments to confirm successful transmission.

File Receiver Documentation

Overview

This document describes the **FileReceiver** class, which listens for incoming file transmissions over a TCP socket. It ensures ordered and reliable receipt of packets and writes received data to a local file.

Components

1. **FileReceiver**: A server that listens for file transmissions.
2. **Packet Handling**: Ensures data integrity and correct ordering.
3. **Threaded Handling**: Supports multiple file transmissions concurrently.

FileReceiver Class

- **Fields**:
  - `port`: Port on which the server listens.
  - `filename`: Name of the file where received data is written.
  - `server_socket`: TCP socket for receiving file data.
  - `expected_seq_num`: Tracks expected packet sequence number.
- **Methods**:
  - `handle_client(client_socket)`: Receives data from a client and writes it to a file.

- `receive_packet(packet)`: Validates received packets and detects corruption or order issues.
- `start()`: Starts the server and waits for connections.

Execution Flow

1. The **server** starts and listens for incoming connections.
2. Upon a connection, it spawns a new thread to handle the client.
3. The client sends **packets** of file data.
4. The server checks for corruption and correct order.
5. Valid packets are written to the file, and **ACKs** are sent.
6. Process continues until the file is fully received.

Running the Program

Execute the script to start the file receiver:

python file_receiver.py

The server will listen on the specified port and save incoming file data.

Summary

The **FileReceiver** ensures reliable file transfer by verifying packets, handling corruption, and maintaining order. It uses multithreading to manage multiple file transfers concurrently.

File Sender Documentation

Overview

The **FileSender** class is responsible for sending a file over a TCP socket to a receiver while ensuring reliable delivery through acknowledgments and retransmissions.

Components

1. **FileSender**: The client responsible for transmitting a file.

2. **Packet Handling**: Sends file data as packets, tracks acknowledgments, and retransmits if necessary.
3. **Threaded Handling**: Uses multithreading to handle acknowledgments concurrently.

FileSender Class

- **Fields**:
    - `server_ip`: IP address of the receiving server.
    - `server_port`: Port number of the receiver.
    - `filename`: Name of the local file to send.
    - `socket`: TCP socket for transmission.
    - `sequence_number`: Tracks the order of packets sent.
    - `sent_packets`: Dictionary storing sent packets for potential retransmission.
    - `max_retries`: Maximum number of retransmission attempts.
    - `timeout`: Timeout before retransmitting a packet.
    - `ack_queue`: Queue to store received acknowledgments.
- **Methods**:
    - `send_packet(data_to_send)`: Sends a data packet to the server.
    - `receive_ack()`: Continuously waits for acknowledgments and handles them.
    - `receive_ack_packet(ack_seq)`: Removes acknowledged packets from the tracking dictionary.
    - `timeout_retransmit()`: Retransmits packets that were not acknowledged within the timeout period.
    - `send_file()`: Reads the file, splits it into packets, and sends them sequentially.
    - `start()`: Initiates a thread for acknowledgment handling and starts file transmission.

Execution Flow

1. The **client** establishes a connection with the receiver.
2. The file is read and split into **packets**.
3. Each packet is sent over the socket with a **sequence number**.
4. The receiver sends **ACKs** for successfully received packets.
5. If an ACK is not received within the timeout, the packet is retransmitted.
6. Once all packets are successfully acknowledged, the transfer is complete.

Running the Program

Execute the script to send a file:

python file_sender.py

The sender will connect to the specified server and transfer the file in a reliable manner.

Summary

The **FileSender** ensures reliable data transmission by verifying packet integrity, handling missing acknowledgments, and retransmitting lost packets. The use of multithreading allows simultaneous acknowledgment processing and file transmission.