Developing a Reliable Data Transfer Protocol

Vibudh Bhardwaj

Mar 20, 2025

CONTENTS:

CHAPTER

ONE

RELIABLE DATA TRANSFER (RDT) PROTOCOL DOCUMENTATION

1.1 Overview

The Reliable Data Transfer (RDT) protocol implementation ensures reliable transmission of data over UDP, providing mechanisms such as sequence numbering, acknowledgment packets, checksums for integrity verification, retransmissions, and a sliding window approach for efficiency.

1.2 Requirements

- Python 3.7+
- Standard libraries: socket, threading, struct, hashlib

1.3 Usage

1.3.1 client.py

```
python client.py <file_to_send> <client_port> <simulator_port>
```

1.3.2 server.py

```
python server.py <received_filename> <server_port>
```

1.3.3 simulator.py

```
python simulator.py
```

1.4 Command-Line Arguments

1.4.1 client.py

Argument	Description
`<file_to_send>`	File path to the data to be sent to the server.
`<client_port>`	Local UDP port to bind the client socket.
`<simulator_port>`	UDP port where the simulator is listening.

1.4.2 server.py

Argument	Description
`<received_filename>`	Name of the file to save received data.
`<server_port>`	Local UDP port where the server listens for incoming packets.

1.5 Functionality

1.5.1 Packet Class

- **Packet(seq_num, data, ack)**: Represents a packet with a sequence number, data payload, acknowledgment flag, and checksum. - **calculate_checksum()**: Generates SHA-256 checksum. - **to_bytes()**: Serializes the packet for sending over UDP. - **from_bytes(bytes_data)**: Deserializes bytes to reconstruct a packet. - **is_valid()**: Verifies packet integrity using checksum.

1.5.2 RDT_Sender Class

- **send(data)**: Sends data reliably, managing a sliding window and retransmissions.
- **_start_timer()**: Starts a retransmission timer for packet timeouts.
- **_timeout_handler()**: Handles retransmissions when packets timeout.
- **_recv_ack_thread(total_chunks)**: Dedicated thread receiving ACK packets to advance the sliding window.
- **close()**: Closes socket and cancels retransmission timer.

1.5.3 RDT_Receiver Class

- **listen()**: Listens continuously for incoming packets, handles acknowledgments, and stores data.
- **_send_ack(seq_num, addr)**: Sends acknowledgment for received packets.
- **get_data()**: Assembles and returns the received data in correct order.
- **close()**: Gracefully terminates the receiver.

1.5.4 NetworkSimulator Class

- Simulates packet loss, corruption, and delays to test the robustness of the RDT implementation.
- Configurable parameters: `loss_rate`, `corruption_rate`, and `delay_rate`.

1.6 Example Usage

1.6.1 1. Start the Simulator

```
python simulator.py
```

Runs the simulator with default parameters (packet loss, delay, corruption).

1.6.2 2. Start the Server

```
python server.py received_file.txt 9001
```

Starts the server to listen on port 9001 and save data as `received_file.txt`.

1.6.3 3. Start the Client

```
python client.py file_to_send.txt 8000 9000
```

Sends `file_to_send.txt` from client port 8000 through simulator at port 9000.

1.7 Notes

- Ensure UDP ports 8000, 9000, and 9001 are open and not blocked by firewalls.
- Adjust simulator parameters to reflect different network conditions.

1.8 License

This implementation is intended for educational and demonstration purposes. Always ensure you have permission for testing network communications.

Author: Vibudh Bhardwaj Last Updated: 2025-03-18