ECS 152A Project 1 Report

Brandon Ng Joon Hoe 924335301
Kevin Shefkiu 924335327

Part 1: iPerf Implementation

1. Introduction

In this part, we developed a UDP client-server data transmission application using Python's socket API. The client sends data ranging from 25 to 200 megabytes to the server. The server then measures and reports the throughput in kilobytes per second. The timestamps, IP addresses, and data sent/received are logged for each run.

2. Implementation

2.1 Server Implementation

The UDP server listens for incoming packets from the client and:

- Waits for a header packet indicating the total payload size.
- Records the timestamp when the first packet is received.
- Accumulates received data in a dictionary, indexed by sequence number.
- Uses sequence numbers to ensure all packets are received correctly.
- Sends acknowledgements (ACKs) for each received packet.
- Detects incomplete transmission and informs the client if data is missing.
- Reassembles the received packets into the original payload.
- Computes throughput as `total_bytes_received / duration` in KB/s.
- Prints relevant details (timestamps, IP addresses, and first 100 bytes of received data).
- Sends the computed throughput back to the client.

2.2 Client Implementation

The UDP client:

- Prompts the user to enter a data size in MB.
- Generates a random payload of the specified size.
- Prints the first 100 bytes of the data along with a timestamp and its IP address.
- Sends a header packet indicating the total payload size.
- Implements a **Sliding Window Protocol**:
    - Sends packets with sequence numbers in a windowed approach.
    - Tracks unacknowledged packets and retransmits them if they timeout.

    - Advances the window when ACKs are received.
- Waits for the server's throughput response and prints it.

3. Development Process

We have experience in implementing TCP communication using socket API to send files before, so setting up the socket API was done with ease. The original implementation successfully established UDP communication for data within the maximum size limit of one packet using the socket API.

Our first implementation struggle was how to handle sending large data which requires sending the multiple packets. Because of this, we could not continue throughput calculation, proper termination signaling, and detailed logging. We used ChatGPT here to provide insights on:

- Using a termination marker (`<<<END_OF_DATA>>>`).
- Measuring time for throughput calculation.
- Improving logging with timestamps and IP addresses.
- Handling socket timeouts and ensuring graceful shutdowns.

We found that using a termination marker can be ineffective in tracking lost and dropped packets. The code was then refactored where the client first sends a header indicating payload size, and data is sent in chunks. The server handles the termination when the size of data received equal payload size. It is here we faced our biggest implementation struggle, which is a large amount of packet drops and no retransmission resulting in lost data.

To address the packet lost, our first solution was a simple Selective Repeat ARQ. On top of the header, the client sends a sequence number with each chunk. After the first reception on the server, the server sends back a message listing missing packet numbers. However, this failed because the number of missing packets was too large to fit into one packet to be sent back to the client.

Our final solution involves using a sliding-window protocol. The client sends a limited number of packets at a time for flow control and preventing large numbers of packet loss. Each packet is retransmitted if its ACK is not received within a short timeout. The server sends back an ACK upon receiving each packet. This addresses the problems faced in our previous implementation and successfully transmit all the data.

Inside Part 1 folder, we separated the implementation to before any ChatGPT is used and our final solution. The client and server file is named appropriately.

4. ChatGPT Interaction Link

https://chatgpt.com/share/67b4ed24-4674-800d-9ea2-2e2229c16b27
