

# COMP 7005

## Project 1

### Report

Brandon Rada  
A01345707  
Dec 2nd, 2025

<b>Purpose</b>	<b>3</b>
<b>Requirements</b>	<b>3</b>
<b>Platforms</b>	<b>5</b>
<b>Language</b>	<b>5</b>
<b>Documents</b>	<b>5</b>

# Purpose

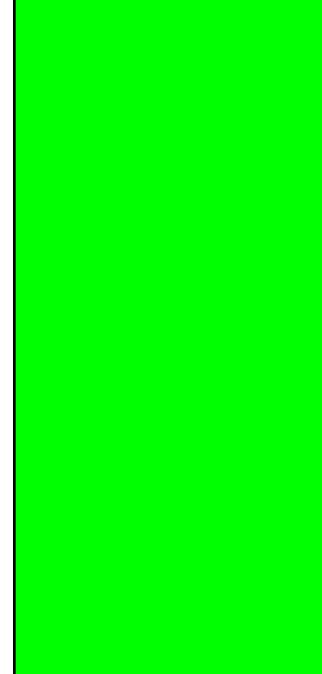
Understand the limitations of UDP and the need for reliability mechanisms. Design a message-based protocol with identifiers and acknowledgments. Implement and evaluate retry logic, timeout handling, and error cases. Simulate packet loss and delay using a configurable proxy server. Measure and describe how the system performs under degraded conditions. Implement reliable communication over UDP by simulating network unreliability and developing a simple protocol that utilizes retransmissions and acknowledgments.

Three programs: a client, a server, and a proxy server that introduces packet loss and delay.

# Requirements

Task	Status
Client	
The client reads and sends messages from standard input to the UDP server.	Fully implemented
It implements a reliability mechanism as follows: <ul style="list-style-type: none"><li>○ Assigns a sequence number to each message.</li><li>○ Sends the message to the server and waits for an acknowledgment.</li><li>○ The client retransmits the message if no acknowledgment is received within the timeout period.</li><li>○ After a maximum number of retries (e.g., 5), the client gives up on that message and prints an error.</li></ul>	Fully implemented
The client supports the following command-line arguments: <ul style="list-style-type: none"><li>○ --target-ip IP address of the server</li><li>○ --target-port Port number of the server</li><li>○ --timeout Timeout (in seconds) for waiting for acknowledgments</li><li>○ --max-retries Maximum number of retries per message</li></ul>	Fully implemented
The client does not attempt to communicate with more than one server and does not implement any connection or handshake logic.	Fully implemented
Server	
The server listens on a UDP socket and receives messages from a client.	Fully implemented

For each valid message: <ul style="list-style-type: none"><li>○ It prints the message to standard output.</li><li>○ It returns an acknowledgment (including the original sequence number) to the client.</li></ul>	Fully implemented
The server does not respond to duplicate messages or out-of-order delivery; it simply acknowledges and displays what it receives.	Fully implemented
It supports the following arguments: <ul style="list-style-type: none"><li>○ --listen-ip IP address to bind to</li><li>○ --listen-port UDP port to listen on</li></ul>	Fully implemented
The server only handles one client at a time and is not required to support concurrent connections.	Fully implemented
<b>Proxy</b>	
The proxy server sits between the client and the server. It forwards UDP packets in both directions while simulating unreliable network conditions.	Fully implemented
It is responsible for: <ul style="list-style-type: none"><li>○ Listening for packets from the client on a specified IP and port.</li><li>○ Forwarding those packets to the actual server address.</li><li>○ Listening for packets from the server and forwarding them back to the client.</li><li>○ Randomly dropping packets based on configured drop probabilities.</li><li>○ Randomly delaying packets based on configured delay probabilities and delay ranges.</li></ul>	Fully implemented
The proxy must support independent configuration for each direction (client-to-server and server-to-client).	Fully implemented
Delay times must be specified as a millisecond range, using minimum and maximum values.	Fully implemented
The proxy supports the following arguments: <ul style="list-style-type: none"><li>○ --listen-ip</li><li>○ --listen-port</li></ul> IP address to bind for client packets Port to listen on for client packets	Fully implemented

<ul style="list-style-type: none"> <li>○ --target-ip</li> <li>○ --target-port</li> <li>○ --client-drop</li> <li>○ --server-drop</li> <li>○ --client-delay</li> <li>○ --server-delay</li> </ul> <p>Server IP address to forward packets to</p> <p>Server port number</p> <p>Drop chance (%) for packets from client</p> <p>Drop chance (%) for packets from server</p> <p>Delay chance (%) for packets from client</p> <p>Delay chance (%) for packets from server</p> <ul style="list-style-type: none"> <li>○ --client-delay-time-min Minimum delay time (ms) for client packets</li> <li>○ --client-delay-time-max Maximum delay time (ms) for client packets</li> <li>○ --server-delay-time-min Minimum delay time (ms) for server packets</li> <li>○ --server-delay-time-max Maximum delay time (ms) for server packets</li> </ul>	
--	--

## Platforms

dc\_shell has been tested on:

- Manjaro
- Ubuntu
- Fedora

## Language

- Python3

## Documents

- [Design](#)
- [Testing](#)
- [User Guide](#)

EXTRA:

In the same folder the 3 programs are run, there is an extra program called clear\_logs.py.  
Run it with python3 clear\_logs.py to clear all 3 log files of their contents.

I made this as a tool to help quickly move onto the next test without manually clearing the contents of each file.