Contents

Network Design Project – Team Virtual (Temp Team Name)

Overview

This repository implements a UDP/TCP-based file transfer protocol across **6 phases**, progressively adding reliability mechanisms and performance evaluation.

Team

Name	Email	Primary responsibility
Zachary Garnes	Zachary_Garnes@student.uml.edu	The whole programming phase (individual submission)

Demo Video (submission)

- **Private YouTube link:** *https://youtu.be/NAMY4LYGDao*

- **Timestamped outline:** *(00:00 → Phase1(a), 00:23 → Phase1(b))*

Repository Structure (required)

Your repo must match this layout (minimum):

```
src/        # sender, receiver, protocol utilities
scripts/    # experiment runner, plotting utilities
docs/       # design documents and diagrams
results/    # CSV + plots generated from experiments
README.md
```

Requirements

- Language/runtime: (Python 3.x)
- OS tested: (macOS / Windows / Linux)
- Dependencies:
  - Python: `pip install -r requirements.txt`

Standard CLI Interface (required)

Your program must support these standardized flags so the TA can run and grade consistently.

Receiver (required flags)

- `--port <int>`: UDP port to bind
- `--out <path>`: output file path to write received bytes
- `--seed <int>`: RNG seed (default: 0)
- `--log-level <debug|info|warning|error>` (default: info)

Sender (required flags)

- `--host <ip/hostname>`: receiver host
- `--port <int>`: receiver port
- `--file <path>`: input file to send
- `--seed <int>`: RNG seed (default: 0)
- `--log-level <debug|info|warning|error>` (default: info)

Injection flags (if required by phase)

- `--data-error-rate <float 0..1>` (default: 0)
- `--ack-error-rate <float 0..1>` (default: 0)
- `--data-loss-rate <float 0..1>` (default: 0)

- `--ack-loss-rate <float 0..1>` (default: 0)

Timing / windowing flags (if required by phase)

- `--timeout-ms <int>` (default: 40)
- `--window-size <int>` (default: 10)

Notes - "Rates" are probabilities per packet/ACK. - Timing experiments must disable verbose logging (use `--log-level warning` or `error`).

Quick Start (Run Locally)

Start Receiver

```
python src/receiver.py --port 9000 --out results/received.bin --seed 0
```

Run Sender

```
python src/sender.py --host 127.0.0.1 --port 9000 --file data/sample.jpg --seed 0
```

Required Demo Scenarios (Current Phase)

Provide the exact commands used to demonstrate each required scenario.

Scenario 1: Send a message (for example, "HELLO") from the UDP client to the UDP server.

Echo the message back from the UDP server to the UDP client. Receiver: Start

```
python udp_server.py
```

Sender:

```
python udp_client.py
```

Expected behavior: - Client sends the text to server, server receives message and send it back to client.

Scenario 2: Transfer a file (for example, a BMP) between a UDP client process and a UDP server process.

```
python rdt1_server.py
```

Sender:

```
python rdt1_client.py
```

Expected behavior: - Client takes .bmp file, breaks it up into packets, sends packets one-by-one to server, server stores all packets data, creates new .bmp from received data, should be the same file that was sent.

Figures / Plots (if required by phase)

Reproduce experiment runs

Output file of the received .bmp by the server

Example:

`recieved.bmp`

Results files

recieved.bmp

Known Issues / Limitations

List any limitations honestly.

Academic Integrity / External Tools

Included packet send print statements to see if any packets were lost