Network Design Project - Phase Proposal & Design Document (Phase 1 of 5)

Team Name: N/A (I just make a group, did not name it)
Members: song ni, song_ni@student.uml.edu (individual work)

GitHub Repo URL: N/A(I will built it later.)
Phase: 1
Submission Date: January 27, 2026
Version: v1

---

0) Executive Summary

This phase implements a basic Reliable Data Transfer (RDT) protocol, version 1.0, over UDP. The primary goal is to establish functional socket communication and a packetization framework for binary file transfer. "Done" is defined as successfully echoing strings in Phase 1(a) and achieving a byte-for-byte identical transfer of a BMP file in Phase 1(b). Validation will be performed by comparing MD5 hashes of the source and destination files and demonstrating the CLI interface via a screen recording.

---

1) Phase Requirements

1.1 Demo deliverable

- **Private YouTube link:** (To be provided at submission)
- **Timestamped outline:**
  - 00:00 Phase 1(a): UDP Socket Message & Echo
  - 01:30 Phase 1(b): RDT 1.0 File Transfer (BMP)

1.2 Required demo scenarios

Scenario	What you will inject/configure	Expected observable behavior	What we will see in the video
1: UDP Echo	Client sends "HELLO" message	Server echoes same message back	Terminal output showing sent/received strings
2: RDT 1.0 Transfer	Path to `sample.bmp` file	Packetization and ordered reassembly	Successful MD5/Diff check of the files

1.3 Required figures / plots

- **N/A**

---

2) Phase plan

2.1 Scope

- **New behaviors added:** UDP socket binding, fixed-size packetization (1024B), and EOF signaling.
- **Behaviors unchanged:** N/A (Initial Phase).
- **Out of scope:** Handling packet loss, bit-errors, or out-of-order delivery (RDT 2.0+ features).

2.2 Acceptance criteria

- ☐ Sender/receiver run with standard CLI flags (`--port`, `--host`, `--file`, `--out`)
- ☐ Phase 1(a) demonstrates bidirectional communication
- ☐ File transfer produces a byte-for-byte identical output
- ☐ All packets are sent/received one at a time (non-pipelined)

2.3 Work breakdown

- **Workstream A:** Socket initialization and Phase 1(a) Echo implementation.
- **Workstream B:** RDT 1.0 packetization logic and binary file I/O.
- **Workstream C:** Documentation and GitHub repository management.

---

3) Architecture + state diagrams

3.1 How to evolve the provided state diagram

For Phase 1, we implement the simplest Finite State Machine (FSM) where the channel is assumed to be 100% reliable. * **Sender:** Remains in a single state "Wait for call from above". * **Receiver:** Remains in a single state "Wait for call from below".

3.2 Component responsibilities

- **Sender:** Reads file, invokes `make_pkt()`, and uses `udt_send()` to transmit.
- **Receiver:** Uses `rdt_rcv()` to capture packets, extracts data, and delivers to the file system.
- **Shared Modules:** `packet.py` for `struct`-based serialization and `util.py` for logging/CLI parsing.

3.3 Message flow overview

`[Binary File] -> Sender (Packetization) -> UDP (Reliable) -> Receiver (Reassembly) -> [Output File]`

---

4) Packet format

4.1 Packet types

- **Type 0:** Data Packet (Contains file content).
- **Type 1:** EOF Packet (Indicates end of file transfer).

4.2 Header fields

Field	Size (Bytes)	Type / Description
Type	1	0 for Data, 1 for EOF
Seq Num	4	Sequence number (Uint32)
Length	2	Payload size (Uint16, 0-1024)
Payload	1024	Binary data chunk

---

5) Data structures + module map

5.1 Key data structures

- **Packet Class:** Stores header fields and byte-array payload.
- **CLI Args:** Dictionary/Namespace for storing ports and file paths.

5.2 Module map + dependencies

- `src/sender.py` -> `src/packet.py`, `argparse`
- `src/receiver.py` -> `src/packet.py`, `argparse`

---

6) Protocol logic

6.1 Sender behavior

1. Initialize socket.
2. Read binary file in 1024-byte chunks.
3. For each chunk: `make_pkt(data)` -> `udt_send(packet)`.
4. Send EOF packet (Type 1) to signal termination.

6.2 Receiver behavior

1. Bind socket and open output file in `wb` mode.
2. `rdt_rcv(packet)`: Extract data from payload.
3. Write `Length` bytes to file.
4. If Type == 1: Close file and exit gracefully.

---

8) Edge cases + test plan

8.1 Edge cases

Edge case	Why it matters	Expected behavior
Last packet < 1024B	Prevents file corruption	Receiver writes only `Length` bytes
File not found	Robustness	Sender prints error and exits
Binary file (BMP)	Integrity check	Hash comparison matches perfectly

8.2 Tests

- **Unit Test:** Verify `packet.encode()` and `packet.decode()` return identical data.
- **Integration Test:** Compare `md5sum` of original and received BMP files.

---

9) Repo structure

“‘text src/ sender.py receiver.py packet.py scripts/ (optional scripts) docs/ DESIGN_DOCUMENT.pdf results/ (test artifacts/screenshots) README.md