# Programming Assignment 2

**CS 436/636 Winter 2024**

Reliable file transfer protocol

**Due Date: March 29, 2024, 11:59PM**

Work on this assignment can be completed individually or in groups of 2

## Objective

The goal of this assignment is to implement a **reliable file transfer** protocol, which could be used to reliably transfer a text file from one host to another **over UDP**. The protocol should be able to handle network errors. For simplicity, your protocol is unidirectional, i.e., data will flow in one direction (from the sender to the receiver) and the acknowledgements (ACKs) in the opposite direction. To implement this protocol, you will write two programs: a sender and a receiver, with the specifications given below.

## Packet Format

All packets exchanged between the sender and the receiver should have the following structure:

```
integer type;          // 0: ACK, 1: Data, 2: EOT
integer seqnum;        // Sequence number of the packet
integer length;        // Length of the String variable 'data'
String data;           // String with Max Length 500
```

The `type` field indicates the type of the packet. It is set to 0 if it is an ACK, 1 if it is a data packet, 2 if it is an end-of-transmission (EOT) packet (see the definition and use of an EOT packet below). For data packets, *`seqnum` is sequence number of the packet*. The sequence number of the first packet should be zero. For ACK packets, `seqnum` is the sequence number of the *packet being acknowledged*. The `length` field specifies the number of characters carried in the data field. It should be in *the range of 0 to 500*. For ACK packets, `length` should be set to zero.

## Sender Program (`sender`)

You should implement a sender program, named `sender`. Its command line input includes the following: `<host address of the receiver>`, `<UDP port number used by the receiver to receive data from the sender>`, `<UDP port number used by the sender to send data and receive ACKs from the receiver>`, `<timeout interval in units of millisecond>`, and `<name of the file to be transferred>` in the given order.

Upon execution, the sender program should be able to read data from a text file (10KB to 15KB) and send the data packet-by-packet to the receiver. After all packets are sent, the sender will start a countdown timer set to the input `<timeout interval in units of millisecond>`. When the sender receives an acknowledgement packet with *seqnum* n, the ACK means that the packet with sequence number n has been correctly received by the receiver. *When a timeout occurs, the sender **retransmits***

***all*** *non-acknowledged packets and restarts the timer*. These steps are repeated until there are no outstanding packets. After all content of the file has been transmitted successfully to the receiver (and corresponding ACKs have been received), the sender should send an EOT packet to the receiver. The EOT packet is in the same format as a regular data packet, except that its *type* field is set to 2 and its *length* is set to zero. The sender can close its connection and exit only after it has received ACKs for all data packets it has sent and received an EOT from the receiver. To keep the implementation simple, *you can assume that the EOT packet never gets lost in the network*.

### *Output*

For both testing and grading purposes, your *sender* program should be able to *generate two log files*, named *seqnum.log,* and *ack.log*. Whenever a packet is sent, its sequence number should be recorded in *seqnum.log*. The file *ack.log* should record the sequence numbers of all the ACK packets that the sender receives during the entire period of transmission. *The format for these two log files is one number per line*. You must follow this format to avoid losing marks.

## Receiver Program (`receiver`)

You should implement the receiver program, named `receiver`. Its command line input includes the following: `<UDP port number used by the receiver to receive data from the sender>`, `<drop probability>` and `<name of the file into which the received data is written>` in the given order.

When receiving packets sent by the sender, it should execute the following:

- if the packet is not EOT, drop the packet with the specified/input probability;
- if the packet is not an EOT and was not dropped, acknowledge the packet; the `seqnum` of the ACK packet should be the same as the `seqnum` of the received packet;
- if the packet is a duplicate (i.e., if it was previously received, not dropped and acknowledged), it should be discarded;
- if the packet is out-of-order, it should be buffered. Received data should be reordered before it is saved in the file.

After the receiver has received all data packets and an EOT from the sender, it should send an EOT packet then exit. Make sure that the file is saved before exiting!

### *Output*

The receiver program is also required to generate two log files, named *arrival.log* and *drop.log.* The file *arrival.log* should record the sequence numbers of all the data packets that the receiver receives during the entire period of transmission. *drop.log* should record the sequence numbers of all the data packets dropped by the receiver. *The format for the log files is one number per line. You must follow the format to avoid losing marks.*

# Hints

- *You must run the programs in the CS Undergrad Environment.*
- Experiment with different sender time-out and receiver drop probability.
- Run `receiver`, and `sender` on 2 different machines *in this order* to obtain meaningful results.
- Make sure all log files and transferred file are saved before exiting. We cannot mark your programs if these files are missing.

## *Example Execution*

1. On the host **host1**: python `receiver.py 9994 0.5 <output File>`
2. On the host **host2**: python `sender.py` **`host1`** `9994 9992 50 <input file>`

# Procedures

## *Due Date*

The assignment is due on **March 29, 2024, 11:59pm**

Late submission policy: 10% penalty every late day, up to 3 late days.

Submissions not accepted beyond 3 late days.

## *Hand in Instructions*

Submit all your files in a single compressed file (.zip, .tar etc.) to LEARN in the dedicated dropbox. The name of the file should include your username(s) and/or student ID(s). You must hand in the following files / documents:

- *Source code* files.
- *README* file: this file ***must*** contain instructions on how to run your program, which undergrad machines your program was built and tested on.
- Text file: If you decide to work in a group, please include a text file that details/lists the members of the group, i.e., their username(s) and/or student ID(s), and names.

Your implementation will be tested on the machines available in the **undergrad environment**.

## *Documentation*

Since there is no external documentation required for this assignment, you are expected to have a reasonable amount of internal code documentation (to help the graders read your code).

You **will** lose marks if your code is unreadable, sloppy, and inefficient.