Assignment 2

Computer Networks (CS 456)

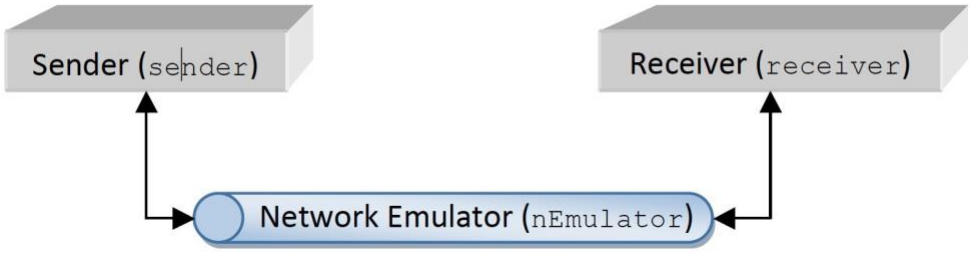
Reliable Data Transfer Protocol over UDP

Due Date: March 11, 2024, at midnight (11:59 PM)

Work on this assignment is to be completed individually

Assignment Objective

The goal of this assignment is to implement **a Reliable Data Transfer** protocol over UDP, which could be used to transfer a text file from one host to another across an unreliable network. The protocol should be able to handle network errors (packet loss), packet reordering, and duplicate packets. For simplicity, your protocol is unidirectional, i.e., data packets will flow in one direction (from the sender to the receiver). Every data packet received by the receiver is acknowledged by the receiver such that the acknowledgement packets (ACKs) will flow from the receiver to the sender. To implement this protocol, you will write two programs: a sender and a receiver, with the specifications given below. To test your implementation, we will provide a third program, the network emulator, that will emulate an unreliable network link.



When the sender needs to send packets to the receiver, it sends them to the network emulator instead of sending them directly to the receiver. The network emulator then forwards the received packets to the receiver. However, it may randomly discard or reorder the received packets. The same scenario happens when the receiver sends ACKs to the sender.

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;                // integer
integer length;        // Length of the String variable 'data'
String data;           // String with Max Length 500
```

Each integer field is a 4-byte unsigned integer in **network byte order**. The `type` field indicates the type of the packet. It is set to 0 if it is a ACK packet, 1 if it is a data packet, 2 if it is an end-of-transmission (EOT)

packet (see the definition and use of an end-of-transmission packet below). For data packets, *seqnum is the 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*. The `data` string should be exactly `length` bytes long. For ACK packets, `length` should be set to zero. A reference implementation of the packet format is provided to you as a Python 3 file named "packet.py".

Sender Program (`sender`)

You should implement a sender program, named `sender`. Its command line input includes the following in the given order:

- `<host address of the network emulator>`,
- `<UDP port number used by the emulator to receive data from the sender>`,
- `<UDP port number used by the sender to receive ACKs from the emulator>`,
- `<timeout interval in units of millisecond>`, and
- `<name of the file to be transferred>`

Upon execution, the sender program should be able to read data from the specified file and send it to the receiver via the network emulator, using the protocol described below.

The sender transmits 10 packets at a time. It starts by transmitting the first 10 packets (or less if the size of the file is less than 5000 bytes). It waits `timeout` milliseconds, then it retransmits all the packets for which no ACK packet was received and, eventually, transmits the following packets, up to a total of 10 packets (e.g., if after sending packets 1, …, 10 and after the timeout the sender realizes that it has received all ACKs but ACK 5, it retransmits packet 5 and transmits packets 11, …, 19). The sender keeps going until all the file has been transmitted and all data packets have been acknowledged. The sender should then 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 socket and exit only after it has received an EOT from the receiver. To keep the project simple, *you can safely assume that the EOT packet never gets lost in the network*.

The first packet of the file must be transmitted with `seqnum 0`, the second packet with `seqnum 1`, and so on.

Output

For both testing and grading purposes, your *sender* program should be able to *generate two log files*, named as *seqnum.log*, *ack.log*. Whenever a packet is sent, its sequence number should be recorded in *seqnum.log* one number per line. The file *ack.log* should record, one number per line, the sequence numbers of all the ACK packets and the EOT packet that the sender receives during the entire period of

transmission (including duplicate ACKs). For EOT packets, the sequence number should be written to the file as "EOT."

Receiver Program (`receiver`)

You should implement the receiver program, named as `receiver`, on a UNIX system. Its command line input includes the following in the given order:

- `<hostname for the network emulator>`,
- `<UDP port number used by the link emulator to receive ACKs from the receiver>`,
- `<UDP port number used by the receiver to receive data from the emulator>`, and
- `<name of the file into which the received data is written>`

When receiving packets sent by the sender via the network emulator, the receiver, who maintains a buffer that can accept more than 10 packets, should execute the following:

- Check the sequence number of the packet.
- If the packet is an EOT packet, send an EOT packet back and terminate the program.
- Else,
    - Send a ACK for the packet
    - If the packet was not previously received, add the packet to the buffer, then, if the received packet follows the last packet written to the file, write the data in the packet and any previously buffered and consecutively numbered packets to the file and remove those packets from the buffer. E.g., Suppose that last packet written to the file has sequence number n and buffer contains packets n+4 and n+2. If received packet has sequence number n+1, buffer it first, write content of packets n+1 and n+2 to the file, then remove these from buffer. Now buffer only contains packet n+4.

Output

The receiver program is also required to generate a log file, named as *arrival.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. *The format for the log file is one number per line or EOT for the EOT packet. You must follow the format to avoid losing marks.*

Network Emulator (`nEmulator`)

The network emulator is provided to you as a Python 3 program. When the emulator receives a data packet from the sender, it will discard it with the specified probability. Otherwise, it stores the packet in its buffer, and later forwards the packet to the receiver with a random amount of delay (less than the specified maximum delay). The same behaviour applies to ACKs received from the receiver. EOT packet from the sender is never discarded. It is forwarded to the receiver once there are no more data packets

in the buffer. EOT packet from the receiver is also never discarded. It is forwarded to the sender once there are no more ACKs in the buffer.

To run `nEmulator`, you need to supply the following command line parameters in the given order:

- `<emulator's receiving UDP port number in the forward (sender) direction>`,
- `<receiver's network address>`,
- `<receiver's receiving UDP port number>`,
- `<emulator's receiving UDP port number in the backward (receiver) direction>`,
- `<sender's network address>`,
- `<sender's receiving UDP port number>`,
- `<maximum delay of the link in units of millisecond>`,
- `<packet discard probability>`,
- `<verbose-mode>` (Boolean: Set to 1, the network emulator will output its internal processing, one per line, e.g. receiving Packet `seqnum` /ACK `seqnum`, discarding Packet `seqnum` /ACK `seqnum`, forwarding Packet `seqnum` /ACK `seqnum`).

Hints

- *You must ensure your programs run in the CS Undergrad Environment*
- *Experiment with network delay values and sender time-out to understand the performance of the protocol.*
- To ensure the programs connect properly, you should run `nEmulator`, `receiver`, and `sender` *in this order*. Please ensure that your implementation works even if the three programs run on separate machines within the CS Undergrad Environment.

Example Execution

1. On the host **host1**: `nEmulator 9991 **host2** 9994 9993 **host3** 9992 1 0.2 0`
2. On the host **host2**: `receiver **host1** 9993 9994 <output File>`
3. On the host **host3**: `sender **host1** 9991 9992 50 <input file>`

Procedures

Due Date

The assignment is due on March 11th, 2024, at midnight (11:59 PM).

Late submission policy: 10% penalty every late day, up to 3 late days. Submissions are not accepted beyond 3 late days.

Hand in Instructions

Submit all your files in a single compressed file (.zip, .tar etc.) using LEARN. The filename should include your username and/or student ID.

You must hand in the following files / documents:

- *Source code* files for your sender and receiver
- *Makefile (if applicable)*: if your program requires compilation, your code **must** compile and link cleanly by typing "*make"* or "*gmake"*
- *README* file: this file ***must*** contain instructions on how to run your program, which undergrad machines your program was built and tested on, and what version of *make* and *compilers* you are using (if applicable).

Your submission may not include any extraneous files (e.g. test files, log files), and the README and Makefile should be in the top-level directory. 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 markers read your code).

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