Assignment 3: Reliable File Transfer over UDP
Due: Oct 19th, Wednesday, 11:59pm
Lead TA: Ali Musa Iftikhar

In this assignment, you will build a simple file transport protocol over UDP.
As UDP offers no reliability, you will have to implement it within your application using the Go-Back-N protocol. Unlike the previous assignment, your server need not handle multiple concurrent clients i.e., it should only handle one file transfer request at a time Also, we expect minimal error handling for this assignment (as is described later). You need to submit the server code, even though for your own testing, you will write a client as well. Submit the server code as a3.c

The protocol is simple and has the following key steps:

1) Bootstrapping: Server will take the port on which it will listen as a command line argument (similar to previous assignments).
2) Request: Client sends a request packet (RRQ) to the server. The request will contain the file name that it wants to fetch, and the window size (for the Go-Back-N protocol) that the server should use.
3) In the typical case, the server starts sending data packets. Each DATA packet will have a sequence number (starting from zero) and will be fixed size (512 bytes). Only the last packet will have a size less than 512 (it could be zero if the file size is a multiple of 512 bytes). This last packet will signal the end of file.
4) For each data packet, the client sends an ACK which carries the sequence number of the corresponding data packet that was received (e.g., data packet 0 will generate ACK 0 and so on)
5) The server uses a fixed timeout value of 3 seconds. If it doesn't receive an ACK, it retransmits the entire window (remember this is Go-Back-N). The client will discard any out-of-order packet that it receives and will also not send an ack for it. So retransmissions from server will only happen due to timeouts. After 5 consecutive timeouts (for the same sequence number), the server should stop the communication.
6) In case the server cannot transfer the file (e.g., file doesn't exist), the server will send an ERROR message (in response to the RRQ message from the client).

Using UDP Sockets

TCP and UDP have many similarities but some key differences too. Some of the things you need to keep in mind are:

- You will no longer use SOCK_STREAM socket type, use SOCK_DGRAM instead.
- You will use recvfrom() and sendto() instead of recv()/read() and send()/write().
- Unlike TCP which may break your message into smaller pieces (or combine smaller pieces into large ones), UDP will preserve message boundaries. So two writes (sendto()) calls on the sender will require two reads (recvfrom()) calls at the receiver. This will actually simplify your message processing (compared to TCP).

Types of Messages

1. Read request (RRQ): type; window size; file name
2. Data (DATA): type, sequence number, data
3. Acknowledgment (ACK): type; sequence number
4. Error (ERROR): type

Assume that these messages will be well formed and you will not need to check for malicious behavior.

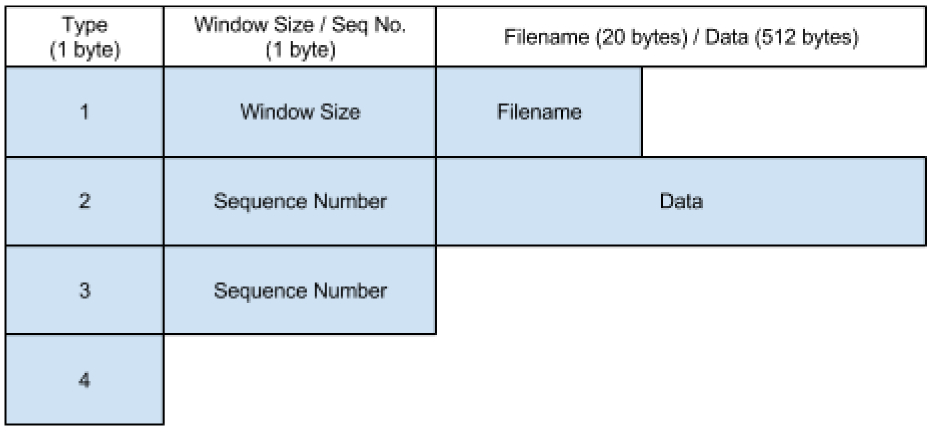
Format

- Type: 1 byte. (RRQ = 1; DATA = 2; ACK = 3; ERROR = 4)
  (e.g., char type = 1; //RRQ)
- Window Size: 1 byte (valid values are 1-9; don't need to worry about any other value)
  (e.g., char win_size = 1)
- Sequence number: 1 byte (valid values are 0 to 50, thus max file size will be limited to this. please don't test files larger than this; it may cause congestion on our network).
  e.g., char seq_no = 40;
- Filename: null terminated string (at most 20 bytes including null).
  e.g., char fileName[20]
- Data: 0 to 512 bytes depending on the packet size.
  e.g., char data[512]

What will we test?

- Basic test (to get any credit): File transfer with window size = 1 (your protocol will behave similar to stop-and-wait)
- File transfer with larger window sizes (2 to 9)
- File transfer with different kinds of ACK losses (e.g., first packet of window gets lost, some middle packet, last packet, persistent loss that lasts more than 5 timeouts, etc)
- Generating error message in case file cannot be transferred (ERROR message).
- We won't test any other scenarios (e.g., some old packet with valid sequence number showing up, or RRQ/ERROR messages getting lost, etc) so you need not worry about them.

Message types



Type (1 byte)	Window Size / Seq No. (1 byte)	Filename (20 bytes) / Data (512 bytes)
1	Window Size	Filename
2	Sequence Number	Data
3	Sequence Number	
4		