A more reliable file transfer on top of UDP

In today's assignment, you need to write a more reliable file transfer program using UDP sockets.

Client program for sending files

The client program can send two types of messages:

- **start** message which initiates the file transfer to a given server
- **data** message which carries part of the file that's being transferred

The client anticipates an ack message for both of the message types.

1. The client program sends a **start** message which initiates the file transfer. Its format is as follows:

   `s | seqno₀ | filename | total_size`

   where
   - **s** indicates that it is **start** message
   - | is a delimiter that divides the neighboring parts in the message
   - **seqno$_0$** indicates the start sequence number for messages of this file transfer session; usually, it equals 0
   - **filename** is the name under which file will be transferred on the server machine, e.g. file1.jpg, resume.txt, etc.
   - **total_size** is the size of the file being transferred in bytes.

2. If the client receives the **ack** in response to the **start** message, it starts transmitting the file by sending data messages one by one. Data message is not longer than the **buf_size.** Message has the following format:

   `d | seqno | data_bytes`

   where

   - **d** indicates that it is a data message
   - **seqno** is the seqno of this data message; if it's a first data message, then seqno=seqno$_0$+1; if it's a second data message then seqno=seqno$_0$+2, etc.
   - **data_bytes** is the part of the file being transmitted. Just raw data

3. When the client receives an ack in response to a **data** message, it will transmit the next data message if there's any left.

4. ~~If the client didn't receive any ack messages 5 seconds after last start or data message send, consider the server to be dead and terminate the program.~~

5. If the client didn't receive ack message for start or data message after 5 retries with 0.5 timeout between them, consider the server to be malfunctioning, terminate the program.

Server program to receive the files

The server can send only **ack** messages. The server should have some data structure (e.g., dictionary)

- to hold the file contents it is currently receiving (from different clients) and
- to record the state of the ongoing sessions, e.g: next_seqno, last reception tstamp, expected size, filename, etc.

Other details of the server operation.

1. If the server receives the **start** message, it replies with an **ack** message which has the following format:

   `a | next_seqno | buf_size`

   where
   - **a** indicates that it is an **ack** message
   - **next_seqno** is the sequence number (seqno) of the next message the server is waiting to receive. In this case, next_seqno equals $seqno_0+1$
   - **buf_size** indicates the maximum size for the data message. Server UDP buffer would be that size.
2. If the server receives the **data** message, it replies with an **ack** message which has the following format:

   `a | next_seqno`

   where
   - **a** indicates that it is an **ack** message
   - **next_seqno** is the sequence number (seqno) of the next message the server is waiting to receive. If it's waiting for n-th data message, then next_seqno equals $seqno_0+n$
3. If the server receives duplicate messages from the client, it should respond with an ack message.
4. If the client isn't active for very long (more than 3s) and the associated file reception session isn't yet finished, then the server should abandon this session and remove everything related to it.
5. The server should hold the information related to the successfully finished file reception for some time (1.0s) before finally removing it.