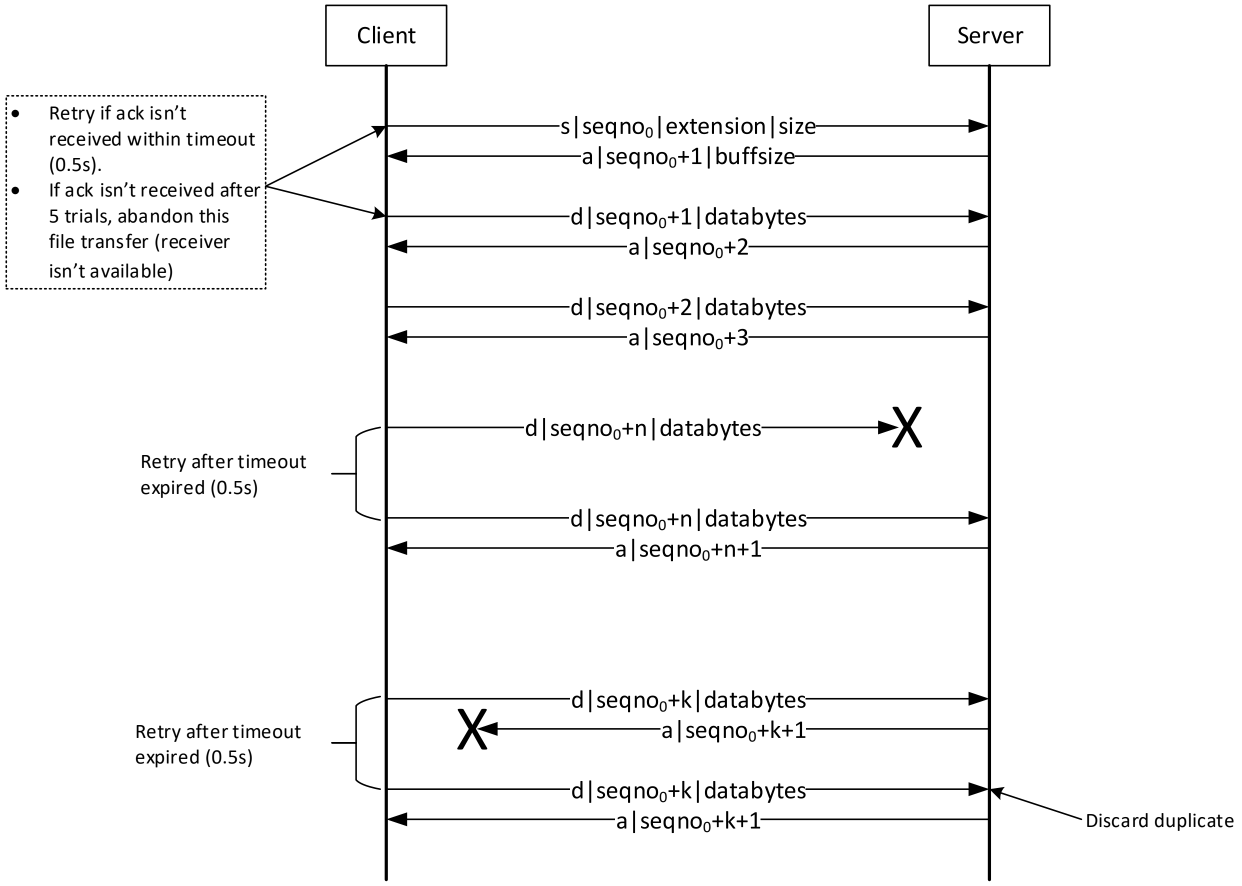
A more reliable file transfer on top of UDP

In an additional assignment for assignment #2, some of you wrote a simple file transfer protocol using UDP sockets. However, that wasn't very reliable. 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 | seqno_0 | extension | 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

- **extension** is the extension of the file that will be transferred, e.g., jpg, txt, doc, py, etc.
- **size** is the size of the file being transferred.

2. If the client receives the **ack** in response to **start** message, it starts transmitting the file in multiple data messages. Data messages are not longer than the **maxsize** and have 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

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

4. If the client doesn't receive the **ack** message within a timeout (0.5s) for both **start** and **data** messages, it retransmits the message and waits for another timeout. If ack isn't received after 5 trials, the client assumes the server is down and gives up the file transfer.

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, extension, 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 | maxsize

   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
   - **maxsize** indicates the maximum size for the message, i.e., the UDP buffer size at the server.
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. The server should ignore the duplicate messages received from the same client

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 this station.
5. The server should hold the information related to the successfully finished file reception for some time (1.0s) before finally removing it.