Kyle Kelly
Prof. Ammar
CS3251
Networking Project 2
21 November 2012

My Protocol:

My implemented protocol is a fairly basic stop-n-wait protocol. I rely on UDP sockets to automatically drop packets that are corrupt, so that is not included. Here are the general steps:

**NOTE: I left out parts of program not dealing directly with transfer protocol.

<u>Sender(Server):</u>

```
1. Wait to receive initial file request.
   a) If valid request, send data packet with seq0, goto 2
   b) Else, goto 1
2. Wait to receive ACK0
   a) If receive ACK0, send next packet with seq1
      i. If next sending packet size < max packet size, goto 4
      ii. Else, goto 3
   b) Else if timeout waiting on response
      i. If tries == MAX_TRIES, goto 1
      ii. Else, tries ++, resend packet with seq0, goto 2
   c) Else, resend packet with seq0, goto 2
3. Wait to receive ACK1
   a) If receive ACK1, send next packet with seq0
   b) If next sending packet size < max packet size, goto 4
      i. Else, goto 2
   c) Else if timeout waiting on response
      i. If tries == MAX_TRIES, goto 1
      ii. Else, tries ++, resend packet with seq0, goto 3
```

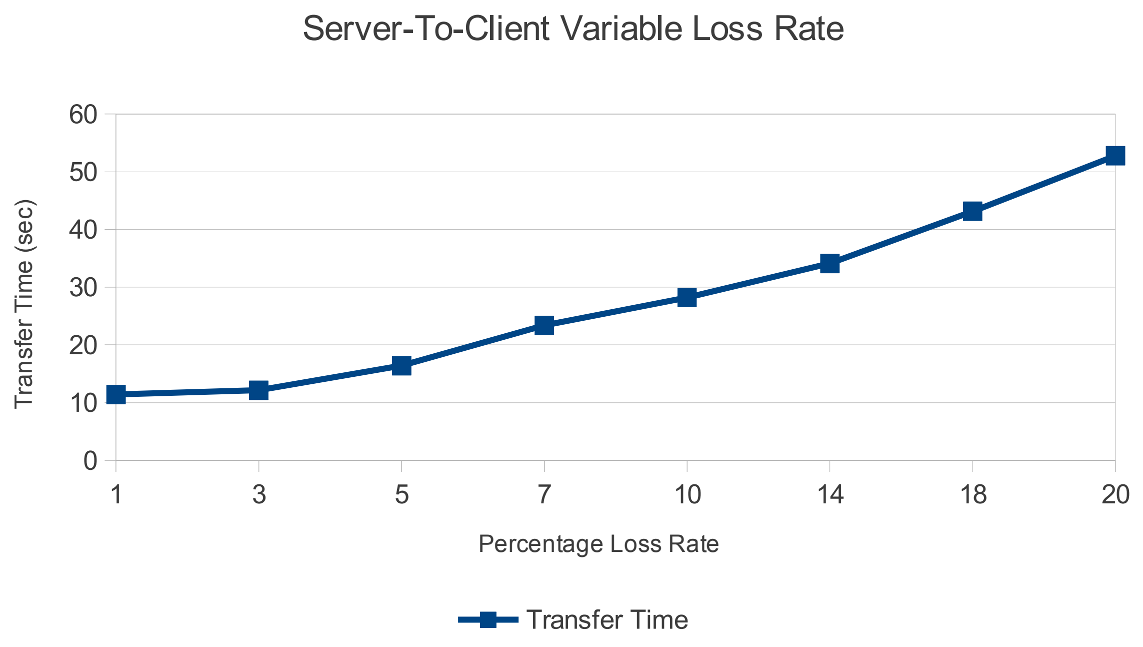
```
    d) Else, resend packet with seq0, goto 3
 4. Send EOF, empty packet with XOR of previous seq num
    a) Goto 5
 5. Wait to confirm termination of file transfer
    a) If recv packet, resend EOF packet, goto 5
    b) Else, tries++
       i. If tries == MAX_TRIES, goto1 (successful transfer)
```

<u>Receiver:</u>

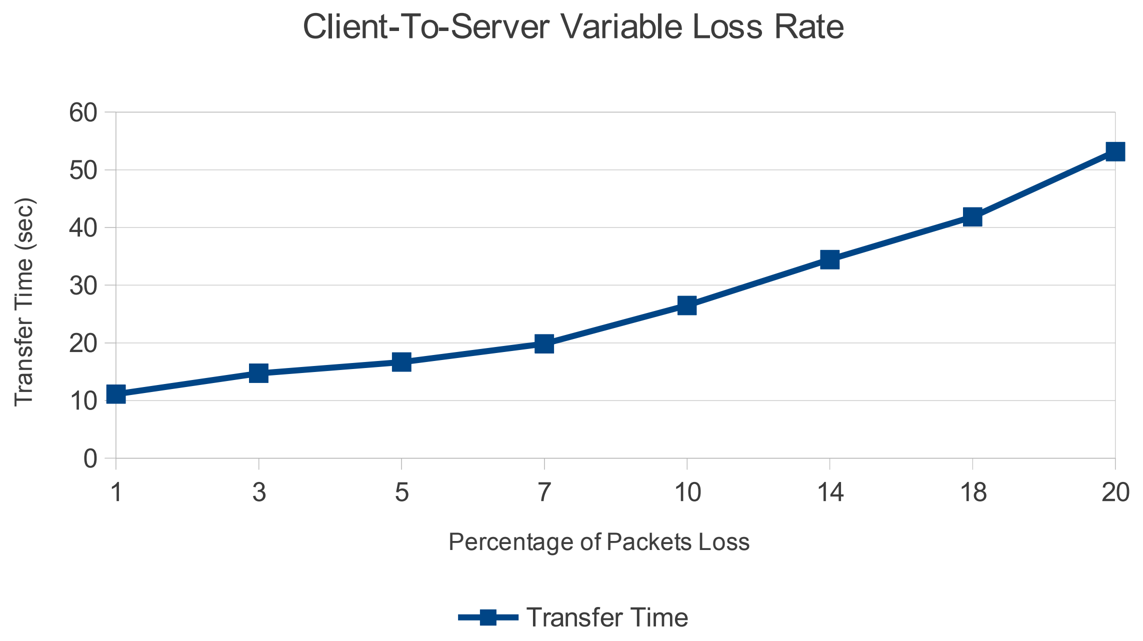
```
1. Send request for file
   a) If timeout waiting for response, goto 1.
   b) If receive first response, ACK seq 0 received, goto 2
2. Waiting for data with seq 1
   a) If data packet received with seq 1, send ACK1, goto 3
   b) If timeout waiting on response, tries++
      1. If tries == MAX_TRIES, terminate
      2. Else, goto 2
   c) Else, send ACK0, goto 2
3. Waiting for data with seq 0
   a) If data packet received with seq 0 , send ACK0, goto 2
   b) If timeout waiting on response, tries++
      1. If tries == MAX_TRIES, terminate
      2. Else, goto 3
   c) Else, send ACK1, goto 3
4. Receive empty EOF packet
   a) Terminate
```

My Statistics:

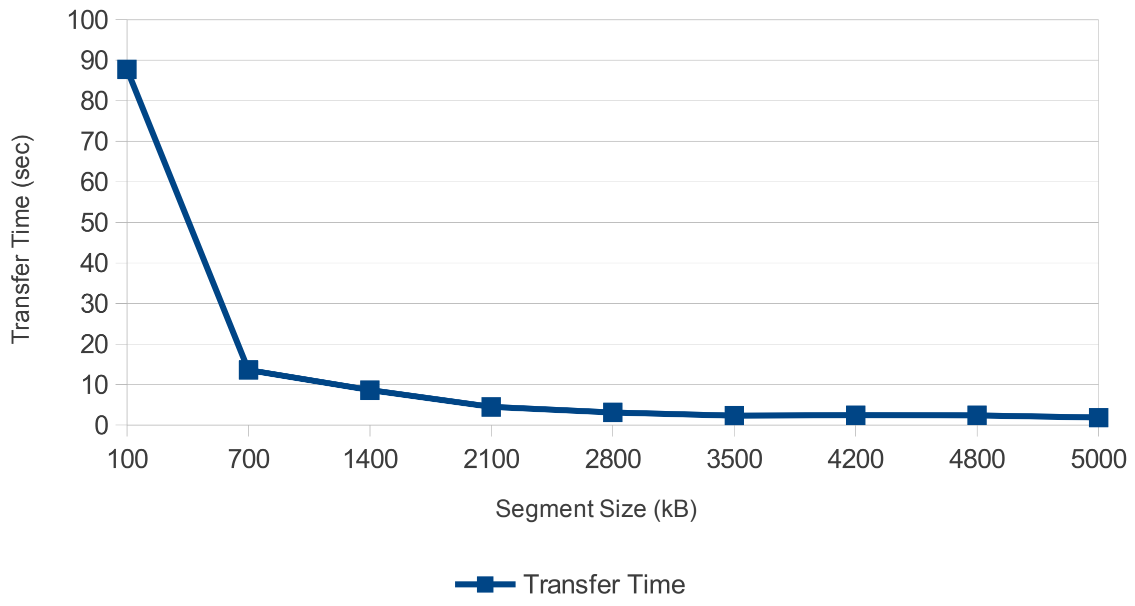
My tests ran all the default control values offered in the project description, except I used 1MB files for control instead of 10MB files, so I testing would not take as long.



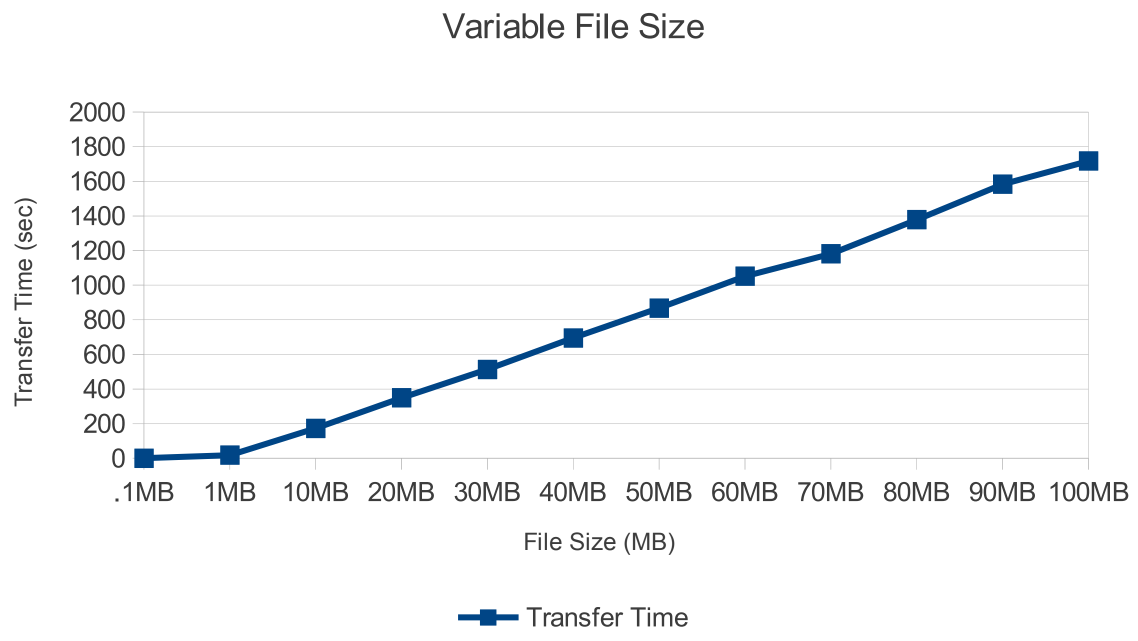
I tested my program using server drop loss rates between 1% and 20%. As you can see, as you linearly increase the packet loss rate from server to client, the transfer time also increases linearly with our protocol. This is because the time spent for each dropped packet is roughly the same, so increasing that linearly causes the same to the time spent transferring.



I tested my program using client drop rates between 1% and 20%The effect is the same when you vary the packet drop/loss rate from the client to the server. The amount of time resending packets is pretty constant, so the linear change in packet loss percentage causes a linear increase in transfer time.



I tested my program with varying segment sizes from 100kB to 5000kB packet segment sizes. As you can see from the above graph, the transfer time is exponentially affected by the segment size. Packet loss time penalties are multiplied by the amount of packets sent by client. Computation per packet is also multiplied by amount of packets required to be sent. One issue not show here, though, is the once packets become big enough, retransmission actually becomes much more costly, so after a time, bigger packets can become an issue. The reason it is not present in this experiment, is because time on the wire for packets in these scenarios was negligible, so the one good effect of smaller packets is lost(retransmission).



I tested my program with variable file sizes from 100kB(.1MB) to 100MB. Variable file size affects transfer time linearly. Basically, as your file size increases, you end up having to break it up and send it in an linearly increasing amount of data packets as well. The amount of dropped packets will average out at each amount, so it doesn't change the rate of transfer time increase.

Program Bugs/Limitations:

In the final testing of my program, I did not encounter any odd bugs, except the occasion seg-fault from accidentally trying to strtok an ACK package in the server when timeouts were occurring. My program is limited to handling one client at a time, and if another client started sending requests in in the middle of a transfer, the results are unkown. To terminate, it also takes a few hundred milliseconds of the server waiting to ensure the client successfully received the termination of file call.