# CS 118 Project 2 Report

Simple Window-based Reliable Data Transfer

Simon Zou - 804347338  Ni Zhang - 104281736

# Implementation Description

We used C's socket library to transfer files between sender and receiver. When the sender gets a request from the receiver, it opens and reads the file up to a defined PACKET_DATA_SIZE (< 1KB), and stores that data in a packet struct.

Our packet struct has the following fields:

- **Type (int)**: Type of the packet
- **Data Size (int)**: Size of the Data field
- **Sequence (sequence)**: Sequence number of the packet
- **Data (char[PACKET_DATA_SIZE])**: Actual data of the file transmission

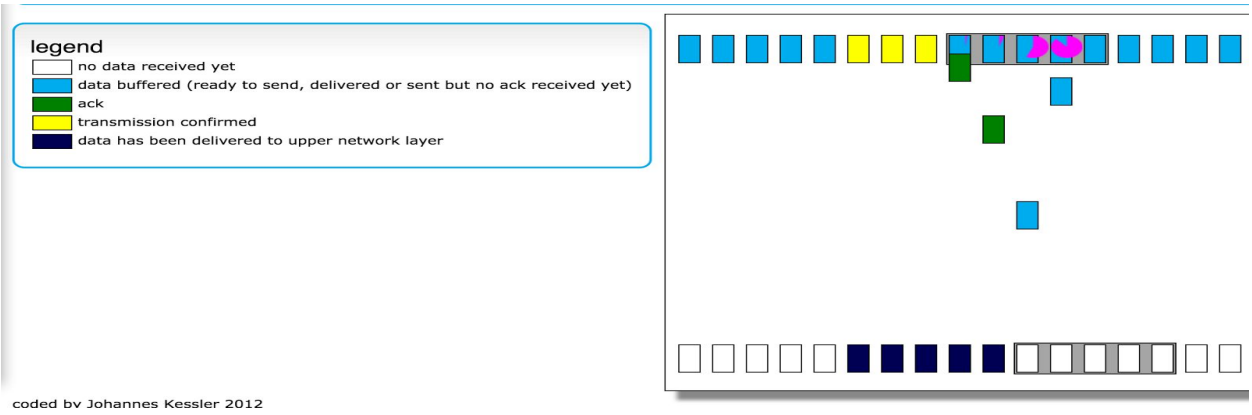
For the type field, we define several types that a packet can be, including

- **ACK** (receiver's acknowledgement that it has got a packet)
- **DATA** (contains the actual data for file transmission)
- **RETRANSMISSION** (for receiver to signal the sender that it needs a retransmission for a corrupted packet)
- **PLACEHOLDER** (contain an expected sequence number, used by receiver to determine where to put out of order packets)
- **FILENAME** (initial packet type for transmitting request).

Based on the window size, packets are made and stored in an array of packets on the sender side and sent. The message passing protocol we use, as defined by the spec, is Selective Repeat. The packets are sent to the receiver, which sends an ACK packet back with the sequence number. Once the sender receives the ACK, if it is not the window's base (i.e. an ACK received out of order), it replaces the original data packet in the window. If it is the base, it will create a new packet from the next part of the file and sends it and moves the window forward for each ACK packet in the array.

The receiver also has a buffer created with PLACEHOLDER packets with the expected sequence numbers. As it receives packets, if the packet sequence is not the base of the buffer (i.e. a packet that arrived out of order), the DATA packet is stored in the buffer replacing a PLACEHOLDER. If it is for the buffer's base, the window similarly slides forward after updating the current base with a new expected number and writing the data in the packet to file and repeats if the new base is a DATA packet.

Each side's "window" only moves forward when it receives a DATA packet or an ACK packet for the base. The following diagram (via http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/) shows how the windows align and ack and data packets being sent.




If the receiver marks a packet as "corrupted"(based on a random number generator that takes in the corruption rate as an input), it sends a RETRANSMISSION packet with a sequence number, which if the sender receives, it will find the DATA packet in its window with that sequence number and sends it again.

If a receiver marks a packet as "lost"(based on a random number generator that takes in the lost rate as an input), it drops the packet and the sender will resend the packet after a defined timeout value. The following screenshot of a sample run shows examples of the different messages and packets being sent.

```
./sender 14000 10 0.0 0.0
Waiting on port 14000
Packet size: 944
 0) Received FILENAME packet, Filename: small.txt
 1) Sent WINDOW_SIZE packet, Window Size: 10, Filesize: 4083, Packets per window: 10
last_packet_index: 4
 2) Sent DATA packet, Sequence: 0, Timestamp: 1457576317
 3) Sent DATA packet, Sequence: 928, Timestamp: 1457576317
 4) Sent DATA packet, Sequence: 1856, Timestamp: 1457576317
 5) Sent DATA packet, Sequence: 2784, Timestamp: 1457576317
 6) Sent DATA packet, Sequence: 3712, Timestamp: 1457576317
 7) Received ACK packet, Sequence: 0,
 8) Received ACK packet, Sequence: 928,
 9) Received ACK packet, Sequence: 2784,
10) Received RETRANSMISSION packet, Sequence: 3712
11) Re-sent DATA packet, Sequence: 3712, Timestamp: 1457576317
12) Received ACK packet, Sequence: 3712,
13) Timeout. Re-sent DATA packet, Sequence: 1856
14) Received ACK packet, Sequence: 1856,
15) Full file size acknowledged: 4083
Time elapsed: 748851
diff small.txt small.txt_copy
➜ CS118-Project2 git:(master) ✗
```

```
./receiver "localhost" 14000 small.txt 0.2 0.1
 0) Sent FILENAME packet, Requested: small.txt
 1) Received WINDOW_SIZE packet, Window Size: 10, Filesize: 4083, Packets Per Window: 10
 2) Created small.txt_copy
 3) Received DATA packet, Size: 928, Sequence: 0
 4) Wrote DATA packet, Sequence: 0
 5) Sent ACK packet, Sequence: 0
 6) Received DATA packet, Size: 928, Sequence: 928
 7) Wrote DATA packet, Sequence: 928
 8) Sent ACK packet, Sequence: 928
 9) Received DATA packet, Size: 928, Sequence: 1856, Lost - Dropping packet
10) Received DATA packet, Size: 928, Sequence: 2784
11) Received DATA out of order packet (Sequence: 2784) put at buffer location 3
12) Sent ACK packet, Sequence: 2784
13) Received DATA packet, Sequence: 3712, Corrupted
14) Sent RETRANSMISSION packet, Sequence: 3712
15) Received DATA packet, Size: 371, Sequence: 3712
16) Received DATA out of order packet (Sequence: 3712) put at buffer location 4
17) Sent ACK packet, Sequence: 3712
18) Received DATA packet, Size: 928, Sequence: 1856
19) Wrote DATA packet, Sequence: 1856
20) Wrote DATA packet from the buffer, Sequence: 2784
21) Wrote DATA packet from the buffer, Sequence: 3712
22) Full filesize received: 4083
23) Sent ACK packet, Sequence: 1856
Time elapsed: 372
```

The sender notifies the receiver of the filesize initially and the receiver closes once the full file is received and written. The sender closes once the full filesize has been acknowledged by the receiver.

# Difficulties and Challenges

We weren't sure initially how the receiver would determine ordering of packets it received but we developed the idea of a placeholder packets to know where to put the packets.

Initially we implemented the protocol without a max sequence number, using the value of ftell(file) as the sequence. When we had to implement a max sequence number this involved changing some code around to use a different method of counting and verification using mod operators.

The edge case of where the initial sender window is bigger than the file size caused problems in that it would cause the sending of nonsense packets or mean the window shouldn't be updated in the same was as normal. Accounting for those meant checking for end of file and using some different indexes.

We found that occasionally the program would hang because the receiver wasn't sending an ACK packet. This was a bug where we immediately wrote the file for an in order packet but forgot to send an ACK.

Figuring out how to implement timeout was tricky. The way we implemented it was a bit hacky and uses gotos to jump in and out of a main while loop. We make a socket non-blocking so that when a packet wasn't received, it jumped to checking the timestamp of the sent packets to see if timeout had occurred (retransmitting if so) and then goes back to checking if a packet was received.