Assignment Report

Language and organization:

The language used to finish the assignment is java, and the codebase is consisting of two java files, Sender.java and Receiver.java. They are built on the sender and receiver given in the programming tutorial, with additional implementations in order to fulfil the assignment's requirements.

Sender Implementation:

The challenging parts of the implementation for me are to handle the wrapping around of the sequence number and the sliding window. The sliding window are mainly dealt by hashmap. Previous sequence number and file offset number is store, so when an ack is received for an earlier segment, the file segment can be retrieved and resent easily. The window size is maintained by keeping global values of the "biggest" ack number received yet and the "biggest" sequence number corresponding to the largest offset number. The left side of the window is maintained by the "biggest" ack number while the right side by the sequence number. The "biggest" number is determined both by relative position and absolute difference, so a smaller number with big difference with its comparator will be considered bigger, if the window size does not exceed 2^15. However, though sliding window is implemented, already acked segments are not deleted because it is difficult to find the sequence number corresponding to the ack number without knowing the data size of that sequence number's packet. Therefore, it will use some extra space and potentially causing confusion when two same sequence number with different data offset number occupy the

same position, though such scenario is very unlikely.

Receiver Implementation:

The challenging part for the receiver is also how to handle the sliding window. An expected sequence number is kept and will only be incremented by the sent data size if the same number is received. If a bigger sequence number is received, it and the data sent is kept in the buffer also implemented with a HashMap. When the expected sequence number is meet, the receiver will begin to write data to the file, and check if the hashmap contains the next expected sequence corresponding to (sequence number + data size)%65536 received. If the hashmap does contain the number, it will continue to write data to the file until the hashmap does not contain the expected number, thus able to write out of order file in order. The receiver will only send ack number to the earliest missing segment and once the missing part is filled out will skip the already received out of order number and send out the newer one due to the implementation above. The data in the hashmap is removed once fetched, so even though the max windows size is not hard code into the hashmap, it should not exceed the max windows size.