Emre Dönmez 21602680
Abidin Alp Kumbasar 21602607
CS421-2

Computer Networks
Programming Assignment 2 Report

Introduction

In this programming assignment, we were asked to implement a selective repeat data transfer algorithm using UDP. We are asked to implement every aspect of the selective repeat protocol using the lecture notes.

In selective repeat, the packages in the current window are sent and the window is shifted when the acknowledgment of the first package in the window is received by the sender. However, different from the go-back-N protocol, the corresponding packages of the received acks in the current window are not retransmitted. Only the packages with no received acks are retransmitted.

In order to achieve this assignment, we are required to write code with concurrency. this mean that we have to use different threads for different packages. We split the to be sent data into bytes of 1022 plus two bytes header and decide the total number of threads as stated in the programming assignment. Also, since the data is divisible by 1022, we had the last package in the size of the residue.

We are asked to repeat the data transfer for 6 different loss rates(the probability that the receiver drops a package) 0, 0.1, 0.2, 0.3, 0.4, and 0.5 with window size 20, maximum receiver delay 100 ms, and package timeout 120 ms. We are also asked to repeat the data transfer with the window sizes 20, 40, 60, 80, and 100 with loss rate 0.1, maximum receiver delay 100 ms, and package timeout 120 ms. at the end of these transfers, the receiver gives us the total time the transmission took in seconds. We are asked to compare the results with different loss rates and different window sizes.

Emre Dönmez 21602680
Abidin Alp Kumbasar 21602607
CS421-2

Results

In order to calculate the throughput (bps), we divide the total transferred bit size to the total time it took to transmit it.

The graphs of total time passes in data transfer and the throughput versus loss rates are the following:

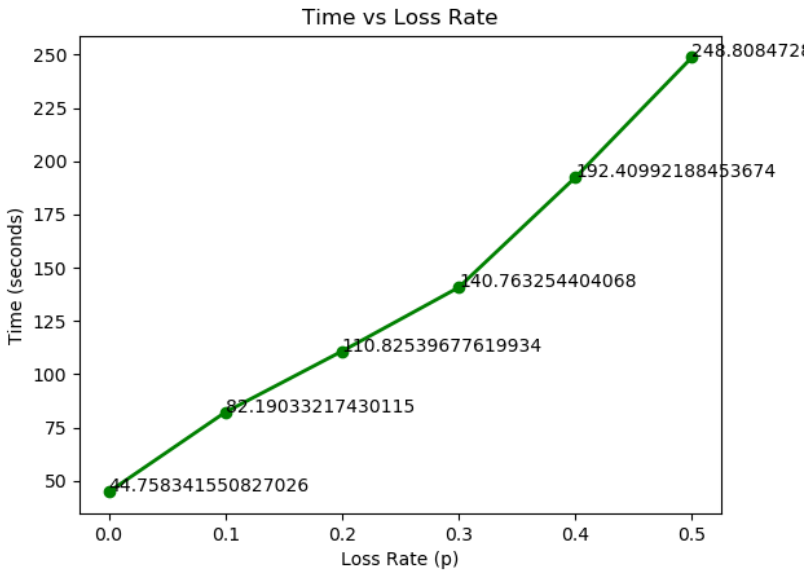

Fig 1: Total Transmission Time versus Loss Rate.

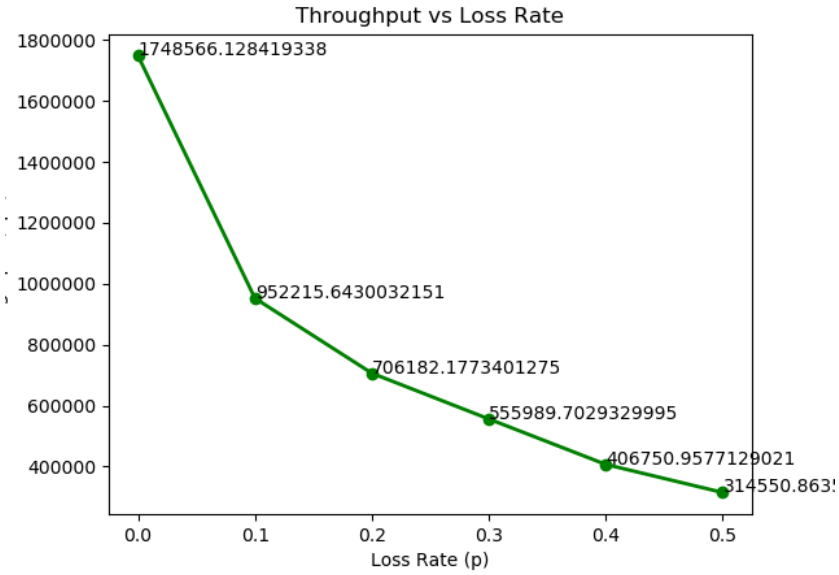

Fig 2: Throughput (bps) versus Loss Rate.

As we can see from the figures, as the loss rate increases, it takes more time to send the data and the throughput decreases. The reason for this is that fact that when the loss rate increases,

more packages has lost acks and t-more packages needs to be retransmitted which causes delays.

The graphs of total time passes in data transfer and the throughput versus window sizes are the following:

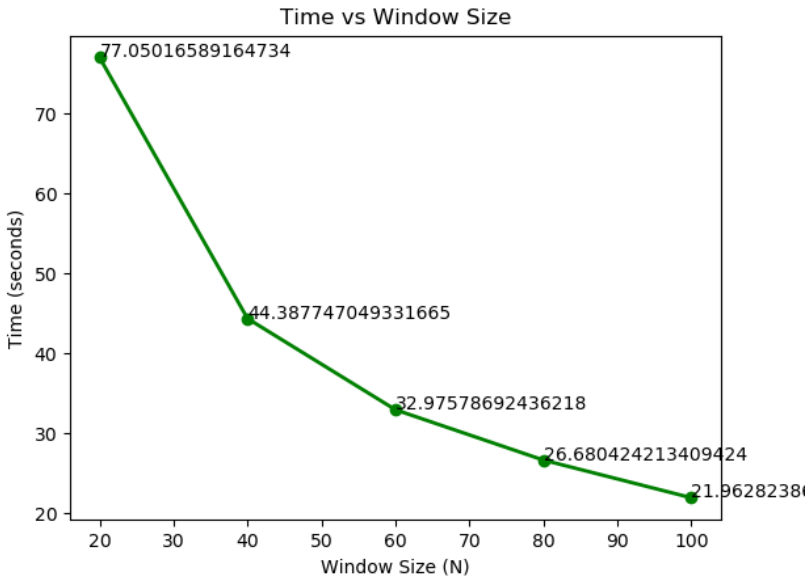

Fig 3: Total Transmission Time versus Window Size.

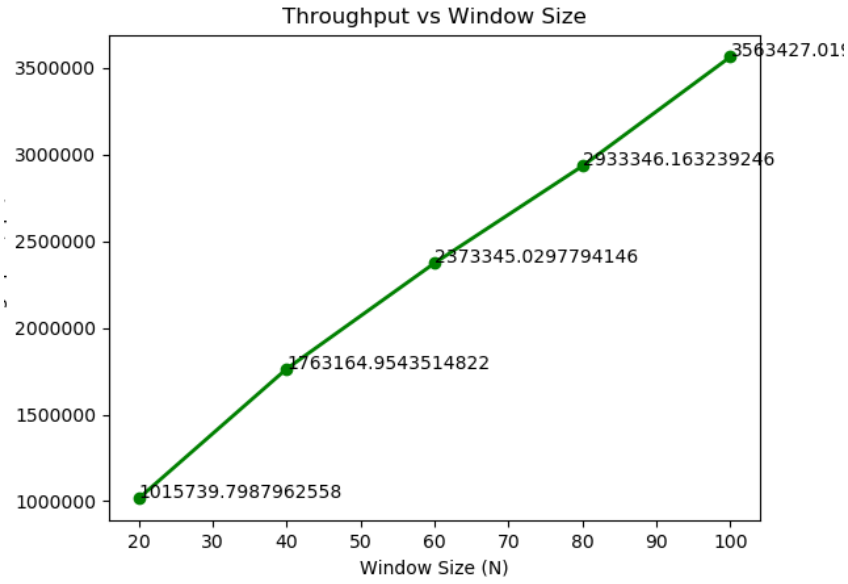

Fig 4: Throughput (bps) versus Window Size.

As seen in the graphs, as the window size increases, the total transmission time decreases and the throughput increases. The reason for this is because more threads are working at the same time transmitting packages in larger window sizes. Since in selective repeat protocol, only the

packages that did not receive acks until their timeouts are retransmitted, more packages are sent and acked in a larger window size compared to a smaller window size.

Conclusion

Overall, we have had a successful transfer. The received image was the same as the sent image and the data size was also the same. We have tried sending different files and it also worked. There was one problem with sending data packets that were too small. If the specified window size was larger than the package number, there was an index error that prevented the threads from starting because we were storing the threads in a thread Array and giving a window size larger than the thread size caused ArrayIndexOutofBounds error. To solve this, we have set the window size to the total package size if it was larger than the total package size. To sum up, this assignment has taught us about the concept of Threading further. Since we are both Electrical and Electronics Engineering students, we have not encountered Threading in any of our previous courses. So the concept was new to us. We have found it to be very useful.

APPENDIX:

Our Code:

```
import java.io.*;
import java.net.*;
import java.util.*;

/**
 * This class communicates with a receiver and send it a
 * file specified in the command line
 * with the specified window size and package timeout.
 * The bind port is also initialized in the command line.
 *
 * @authors  Emre Dönmez, Abidin Alp Kumbasar
 * @version 1.0
 * @since   22.12.2019
 */
public class Sender {

	static InetAddress clientIPAddress;
	static DatagramPacket[] buffer;
	public static int seq = 1;

	public static void main(String[] args) throws Exception, SocketException {

		String filePath = args[0];
		int rec_port = Integer.parseInt(args[1]);
		int window_size = Integer.parseInt(args[2]);
		int trans_timeout = Integer.parseInt(args[3]);
		InetAddress ip = InetAddress.getByName("127.0.0.1");

		int window_start = 1;
		byte[] img = img_reader(filePath);
		int[] pckt_info = packet_no_find(img);

		if(pckt_info[1] + 1 < window_size) {
```

```
                window_size = pckt_info[1];
            }
            Thread[] threads = new Thread[pckt_info[1] + 2];
            DatagramSocket serverSocket = new DatagramSocket(53000);

            byte[] senderData;
            byte[] senderDataLast;
            byte[] receiveData = new byte[2];
            ArrayList<Integer> rec_ack = new ArrayList<>();
            for(int i = 1; i <= pckt_info[1]+1; i++){
                rec_ack.add(i);
            }

            for(int i = 1; i < pckt_info[1] + 2;i++ ) {
                if(i == pckt_info[1] + 1){
                    senderDataLast = packetCreater(pckt_info[0] + 2, img,
i, true, pckt_info[0]);
                    threads[i] = new Thread(new SenderThread(serverSocket,
trans_timeout, senderDataLast, ip, rec_port, i));
                }
                else{
                    senderData = packetCreater(1024, img, i, false,
pckt_info[0]);
                    threads[i] = new Thread(new SenderThread(serverSocket,
trans_timeout, senderData, ip, rec_port, i));
                }
            }

            int ack_counter = 0;
            while(ack_counter < pckt_info[1] + 1) {
                for(int i = window_start; i < window_start + window_size;i++)
{
                    if(threads[i].getState() != Thread.State.NEW) {

                    }
                    else {
                        threads[i].start();
                        //System.out.println(i);
                    }
                }
                DatagramPacket rc = new DatagramPacket(receiveData,
receiveData.length);
                serverSocket.receive(rc);
                int ack = fromByteArray(receiveData);

                if(window_start <= ack && ack < window_start + window_size) {
                    if ( threads[ack].isAlive()) {
                        threads[ack].interrupt();
                        ack_counter++;

                        int remove_index = rec_ack.indexOf(ack);
                        if(remove_index != -1) {
                            rec_ack.remove(remove_index);
                        }

                        if(rec_ack.size() != 0) {
                            window_start = rec_ack.get(0);
                        }
```

```
                                else {
                                        System.out.println("All packets sent.");
                                }

                                if(window_start > pckt_info[1] - window_size + 1)
{
                                        window_start = pckt_info[1] - window_size
+ 2;
                                }
                        }
                }
        }
        byte[] dat = new byte[2];
        dat[0] = (byte)0;
        dat[1] = (byte)0;
        DatagramPacket pk = new DatagramPacket(dat, 2, ip, rec_port);
        serverSocket.send(pk);
        serverSocket.close();
    }

    /**
     * This method reads a specified file into a byte array.
     *
     * @param  filePath is the location of the file to be read.
     * @return bytesArray is the byte array form of read file.
     */
    public static byte[] img_reader(String filePath) throws IOException {
        File file = new File(filePath);
        //init array with file length
        byte[] bytesArray = new byte[(int) file.length()];

        FileInputStream fis = new FileInputStream(file);
        fis.read(bytesArray); //read file into bytes[]
        fis.close();

        return bytesArray;
    }

    /**
     * This method takes a portion of 1024 bytes from a file and creates a
package,
     * depending on the package number to be sent where
     * 1022 bytes are the file data and the 2 bytes is the header.
     * If the package is the last package, it takes the
     * remaining bytes from the file.
     *
     * @param leng is the length of the file in bytes.
     * @param image is the file in byte array format.
     * @param pckt_no is the packet number of the to be created package.
     * @param isLast specifies if the package is the last package or not.
     * @param last_len is the length in bytes of the last package.
     * @return packet is the created package.
     */
    public static byte[] packetCreater(int leng, byte[] image, int pckt_no,
boolean isLast, int last_len){
        byte[] hold = convertToBytes(pckt_no);
        int start = (pckt_no-1)*1022;
        //System.out.println(hold);
```

```
            byte[] packet = new byte[leng];
            packet[0] = hold[1];
            packet[1] = hold[0];
            //System.out.println(packet[1]);//python receiver.py 62000 20 0.1 100
            if(isLast){

                  for(int i = 0; i < last_len +2; i++){
                        if(i<2){
                              //packet[i] = hold[i];
                        }
                        else{
                              packet[i] = image[start];
                              start++;
                        }
                  }
            }

            else{
                  for(int i = 0; i < 1024; i++){
                        if(i<2){
                              //packet[i] = hold[i];
                        }
                        else{
                              packet[i] = image[start];
                              start++;
                        }
                  }
            }
            return packet;
      }

      /**
       * This method converts a byte array to an integer.
       *
       * @param  bytes is the byte array to be converted to an integer.
       * @return is the byte array in integer format.
       */
      public static int fromByteArray(byte[] bytes) {
            return bytes[0] << 8 | (bytes[1] & 0xFF) << 0;
      }

      /**
       * This method converts an integer to a byte array.
       *
       * @param  i is the integer to be converted.
       * @return byteForm is the integer in Byte array format.
       */
      public static byte[] convertToBytes(int i) {
            byte[] byteForm = new byte[2];
            int next = i % 256;
            byteForm[0] = (byte) next;
            next = i / 256;
            byteForm[1] = (byte) next;
            return byteForm;
      }

      /**
       * This method calculates the number of packages to be
```

```
         * sent and the length of the final package.
         *
         * @param  img is the file to be sent.
         * @return out is the array with the last package size
         *  and total package number - 1
         */
        public static int[] packet_no_find(byte[] img){
                int[] out = new int[2];
                int size = img.length;

                int number = size/1022;
                int lastPacket = size - number*1022;

                out[0] = lastPacket;
                out[1] = number;

                return out;
        }
}

/**
 * This class extends Thread and sends specified packages until
 *  they are terminated by the main Thread.
 *
 * @authors  Emre Dönmez, Abidin Alp Kumbasar
 * @version 1.0
 * @since   22.12.2019
 */
class SenderThread extends Thread {

        DatagramSocket socket;
        int timeout;
        byte[] packet;
        InetAddress ip;
        int rec_port;
        int name;

        /**
         * This is a Constructor
         *
         * @param  socket is the connection socket.
         * @param timeout is the package send timeout.
         * @param packet is the packet to be sent in byte array format.
         * @param ip is the ip address of the connection.
         * @param rec_port is the bind port of the connection.
         * @param name is the package number of the Thread.
         *
         */
        SenderThread(DatagramSocket socket, int timeout, byte[] packet, InetAddress
ip, int rec_port, int name){
                this.socket = socket;
                this.timeout = timeout;
                this.packet = packet;
                this.ip = ip;
                this.rec_port = rec_port;
                this.name = name;
        }
```

Emre Dönmez 21602680
Abidin Alp Kumbasar 21602607
CS421-2

```
	/**
	 * This method overrides Threads run function.
	 * It sends the specified package and waits for timeout. This
	 * iterates until the Thread is interrupted by the main Thread.
	 *
	 */
	public void run() {
		DatagramPacket pk = new DatagramPacket(packet, packet.length, ip,
rec_port);
		try {
			while(true) {
				// Send packet
				socket.send(pk);
				// Wait for main thread notification or timeout
				Thread.sleep(timeout);
			}
		}

		// Stop if main thread interrupts this thread
		catch (InterruptedException | IOException e) {
			return;
		}
	}


}
```