TITLE OF THE PROJECT BASED WORK:

Design and Implementation of GBN & SR -TRANSPORT LAYER Protocols, Using a GUI MODEL.

Report submitted to the SASTRA Deemed to be University as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

ARCHANA S

(REG NO:124015011, IT III YEAR)

DECEMBER- 2022

SCHOOL OF COMPUTING

THANJAVUR – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Design and Implementation of GBN & SR-Transport Layer protocols, using a GUI model, their comparison"** submitted as a requirement for the course , **CSE302: COMPUTER NETWORKS** for B.Tech. is a Bonafide record of the work done by **Shrimathi ARCHANA S (Reg. No.124015011, B.TECH ,IT-3RD YEAR) ,** during the academic year 2021-22, in the School of Computing (SOC) .

Project Based Work *Viva voce* held on : Dec 2022

Examiner 1 **Examiner 2**

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

GBN	Go Back -N (Transport Layer Protocol)
SR	Selective Repeat (Transport Layer Protocol)
PKT	used to mention a transport layer segment/packet
RTT	round trip time(between sender and receiver)
ACK	positive acknowledgement(a control packet, sent by receiver ,upon receiving a valid packet)
NACK	negative acknowledgement(a control packet, sent by receiver ,upon receiving a corrupt/invalid packet)
RDT	reliable data transfer

ABSTRACT SUMMARY

The application programmer, needs to be provided with a robust method, to transmit data over, internet, to a receiver, which need not be having a direct link, to sender (Reliable Data Transfer). While ensuring this reliability, one needs to also ensure that the different rates at which sender/receiver process data should not be a bottleneck (Flow control), and also, this reliability must be achieved with minimum possible usage of bandwidth, that is it must not waste/congest a channel (already congested), just to achieve this reliability (Congestion Control). These are the main responsibilities of Transport Layer.

It must be remembered that, underlying network layer (IP), is un-reliable, and thus a transferred packet, has a good probability to be dropped/lost, by a intermediate router, or may be overly delayed, in reaching receiver. Thus a feedback mechanism must be used, between sender-receiver, and it must be ensured that, not receiving a feedback within a particular time, should be treated with packet getting lost.

We must also deal with case, where a packet is correctly received, but is found to be corrupted. We need to signal it to sender, so as to receive it again, correctly (NACK). Whereas we also need to signal to sender, a correct receival of a packet (ACK). Now, the problem will be, how a receiver will know, if a packet is a re-transmission or a new packet, we use sequence numbers, to deal with that case. And so on. The above must be done, efficiently, with minimal wastage of resources(bandwidth).

KEYWORDS:

Reliable Data Transfer,

Transport Layer,

Flow Control,

Congestion Control,

NACK, ACK,

Re-Transmission,

Sequence Numbers.

REFERENCES

• Computer Networking: A Top-down Approach James kurose-W.Ross**(Prescribed textbook, for this course)**

• The Complete Reference JAVA- Herbert Schildt **(Prescribed Textbook for java)**

• docs.oracle.com (for referring documentation of java swing components)

• javatpoint.com (for exploring about, various swing components)

• stackoverflow.com (for programming related solutions)

INDEX

INTRODUCTION

i) PRINCIPLES OF RELIABLE DATA TRANSFER

The content within a packet (transport layer segment) is corrupt or not, can be checked using the **CHECKSUM**. It is a field in every **TCP/UDP** segment structure. When I say , a packet is corrupt, I mean the bits in that packet are flipped from 0 to 1,vice versa ,that is inverted .As already mentioned, it can be easily identified, and if more bits are allotted (other than for storing checksum) , original data can be recovered, from faulty/corrupted packet .

What is meant by a reliable channel is that :

1) No packets are lost, in the underlying network

2) They are delivered ,in the order in which they were sent.(I have mentioned about packet reordering in the section ***"FUTURE PLANS")***

3) No data bits are corrupted OR there exist ways ,to derive original data bits, from corrupted data.

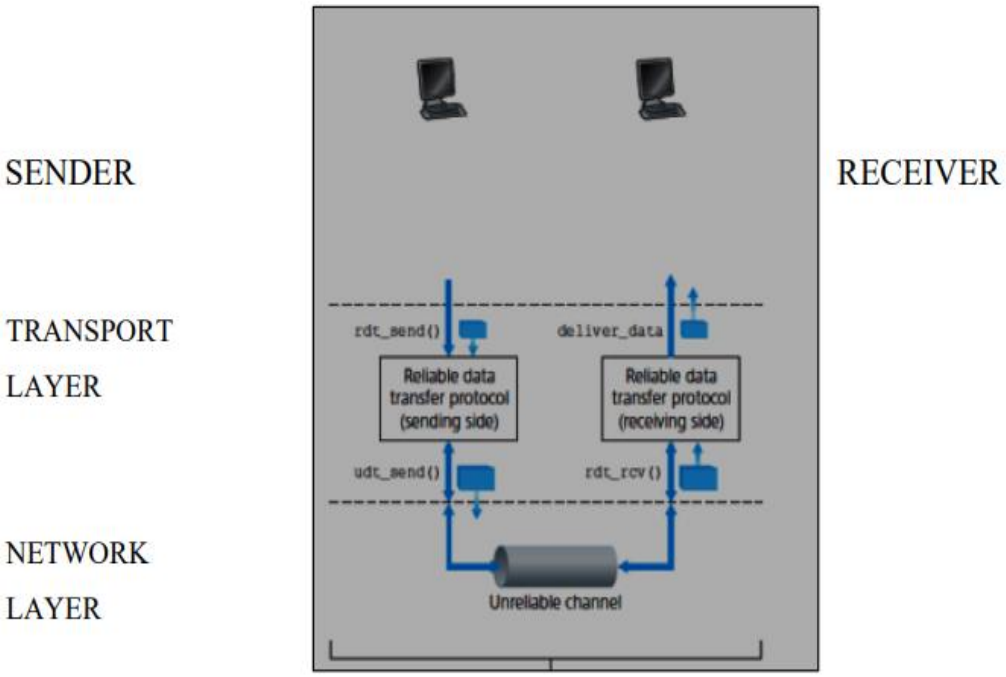
Why is a "reliable channel" a challenge to achieve?

1) The underlying Network layer ,is un-Reliable.

2) The **CONGESTION** in a network, the need to have **FLOW CONTROL** ,results in dropping of some packets ,by intermediate routers.

3) It is not the case that between a sender ,receiver ,there is a single direct physical link ,but its usually a **GLOBAL INTER-NETWORK.**

FIG 1



II) BUILDING A RELIABLE DATA TRANSFER PROTOCOL

Now we will see different types of UNDERLYING CHANNEL, and the methods involved ,in the algorithm of sender-receiver to overcome them ,and maintain reliability:

I)When underlying channel is completely reliable:

A single state **"STOP-AND -WAIT "** protocol is enough .The receiver receives data, at the same pace, at which sender sends data, thus no **FLOW CONTROL** is needed, in this case .

DISADVANTAGES:

Such a underlying network is not possible in reality. It does not deal with **CONGESTION CONTROL,** it assumes that **FLOW CONTROL**, is not needed, which are only theoretical ideas.

II) When Reliable Data Transfer Over a channel with Bit Errors:

Bit Errors refers to flipping of 0 to 1, vise versa, in the data bits of a packet ,due to its transmission in the underlying network.

A **“stop-and-wait”** Protocol with positive ,negative ACKNOWLEDGEMENT is needed. ***Positive acknowledgement(ACK)*** is sent ,for every correctly received packet, and ***negative acknowledgement(NACK)*** is needed for giving feedback about a packet received with bit errors.

In order to differentiate ,whether the received packet is a **DUPLICATE PACKET**(RE-TRANSMITTED PACKET) or a new packet, we go for **SEQUENCE NUMBERS.** Each packet ,sent by sender carries a sequence number ,with the help of which ,the receiver identifies ,if it is a re-transmission.(every alternate packet sent, has different sequence numbers).

We can avoid using NACK, by sending a ACK for the last correctly received packet, this reduces programming needed in the sender side, and reduces complexity of the algorithm.

Thus we use the following, in this case to achieve reliability:

1)check summing

2)sequence numbers

3)ACK packets

4)re-transmissions(the duplicate packet, sent to compensate for the corrupted packet)

DISADVANTAGES:

If a ACK ,NACK gets lost in a network, then the above protocol will fail. And moreover ,there is **possibility of error within the ACK/NACK,** that is the data of the control packet, getting corrupted .We can add **CHECKSUM** ,and to detect this damage, while sender receives the packet.

III) Reliable Transfer Of Data over a Lossy Channel, with Possibility Of Bit errors:

In this case ,the underlying network can **lose the packets as well,** along with the possibility of bits in data getting flipped/inverted.

Thus now, the ACK by the receiver, or the packet by the sender, can be lost in the network, and our protocol should be able to handle this. There also exists possibility, of a packet **getting Overly Delayed,** due to various reasons(Congestion ,etc.)

The Sender must wait for a certain Time, before concluding that the packet is lost.

Min Time to wait for a packet= RTT(from sender to receiver) + processing time needed by receiver(per packet)

RTT- refers to the **ROUND TRIP TIME**, between 2 hosts.

Thus for all of the following cases:

1)packet is lost

2)an ACK was lost

3)packet or ACK were overly delayed

The solution is same, to RE-TRANSMIT.

A **COUNTDOWN TIMER** is used, to measure the delay. Thus the sender must include hardware to implement it.

<u>DISADVANTAGES:</u>

1) It works in a manner, popularly known as “STOP-AND-WAIT” (protocol), which **FAILS TO UTILISE THE BANDWITH**, when the link is dedicated/free .

2) No scope for **PIPELINED OPERATION.**

3) Sender remains idle, for a majority of the time, which is a wastage for its processing capability(it remains idle till a TIMEOUT happens, or a ACK is received)

III) WAY FORWARD:

To ensure pipelined operations, so that entire free bandwidth can be used ,we must add following specifications:

1) Sender must be allowed to send a particular MAX number of packets, without getting any acknowledgement(without waiting).

2) Range of SEQUENCE NUMBERS must be increased, to ensure each of the packets ,sent by sender is uniquely identified, distinguished from a **RE-TRANSMIT(DUPLICATE PACKET)**

3) **Physical HARDWARE BUFFER** needs to be added into sender/receiver. Sender needs to buffer un-acknowledged packets, receiver needs to buffer those packets ,which are received out of sequence,(once a full sequence of packets is received, it can together be delivered to upper layer).

4) Popular techniques in this direction are GO-BACK-N, SELECIVE REPEAT protocols.

FIG 2:- STOP AND WAIT OPERATION

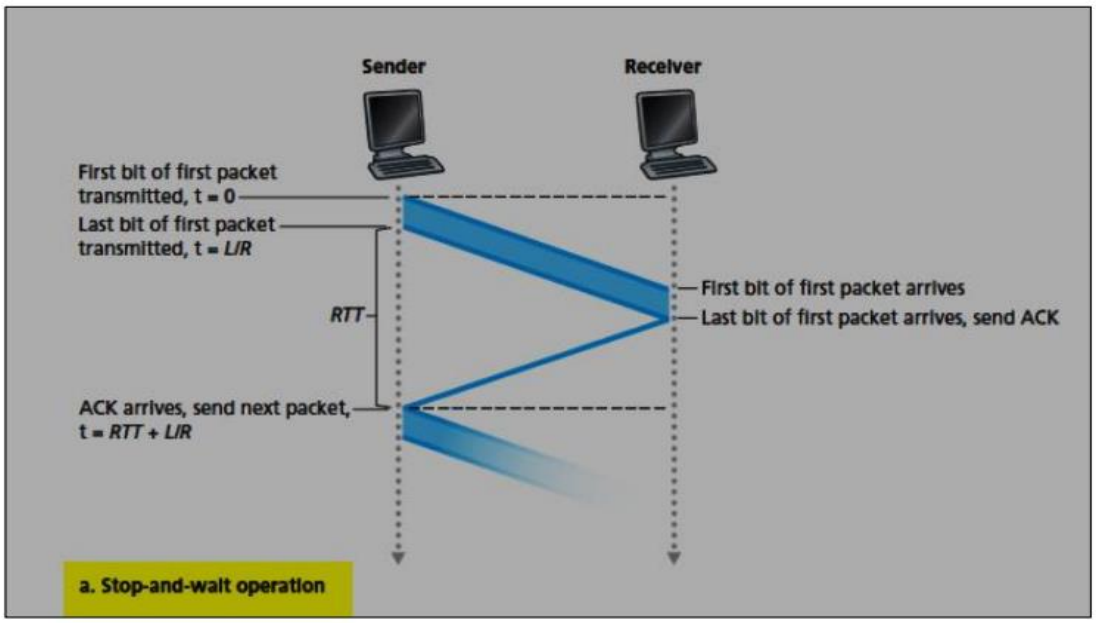
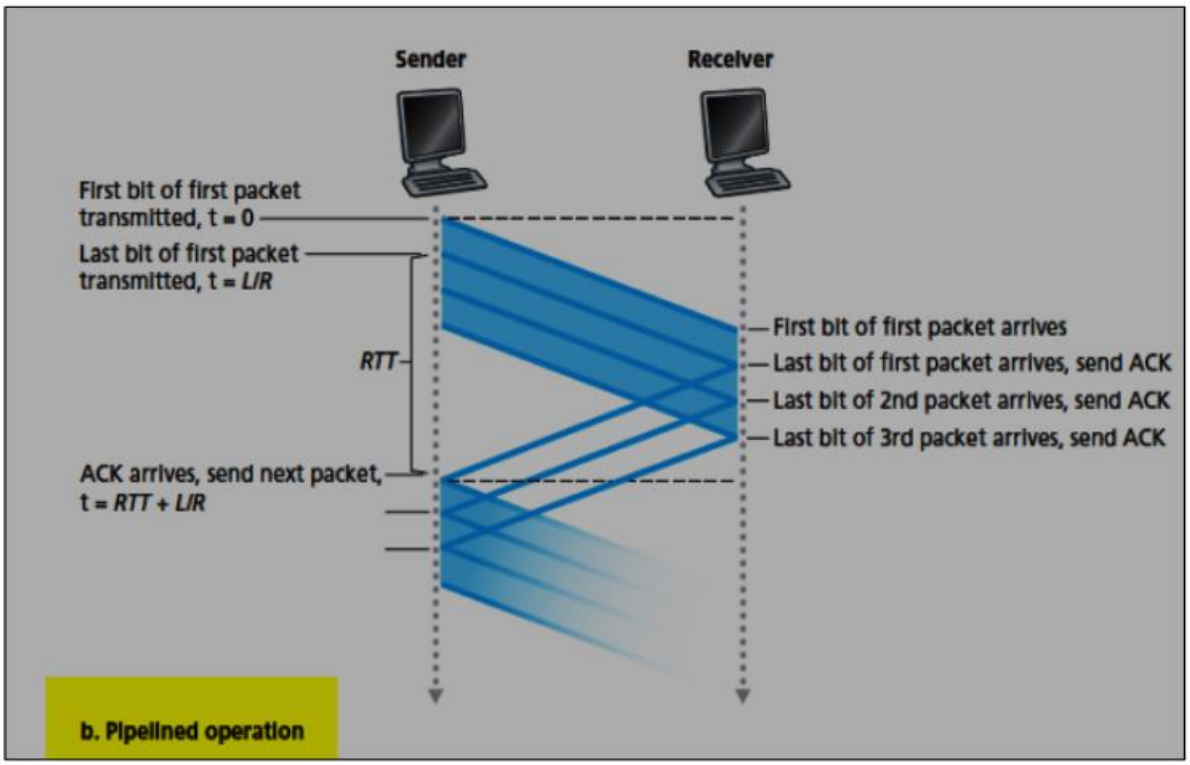


FIG 3: PIPELINED OPERATION



GO BACK -N PROTOCOL(GBN)

➢ The Sender in this protocol, is allowed to send multiple Packets,(when asked by upper layer), without receiving a single ACK, but the maximum number ,of packets, that can be sent, like this is constrained to a number N.

➢ That is, at a time, a max of N unacknowledged packets are allowed in the PIPELINE.

EXPLANATION ABOUT ITS WORKING:

Here is a sample ,of how my application looks, while in transition:-

Sender in GBN:

➢ In this case N=5 ,is set as Default. That is the pipeline can have a Max of 5 unacknowledged packet ,at any time.

➢ Here we have taken **SEQUENCE NUMBER** of range [0-19] ,for simplicity, of GUI.

➢ The packets currently lying inside the sender side window of size=5, are printed with

RED ink. Example : [2,3,4,5,6] in the above picture

➢ The packets with a **YELLOW** background color are acknowledged, by receiver.

➢ The packets with **BLUE** background color, are sent, but the sender is yet to receive ACK for them.

➢ Here the **MAXIMUM WAITING** time, for a packet is taken as 15 seconds(see timer at top-center), for the purpose of simplicity.(maximum waiting time, must be greater than :-

[RTT + processing time by receiver] , in reality.)

➢ The **"base", "next Sequence No"** field, are updated with each new packet sent, ACK received.

➢ The packets with **a WHITE** background are, not yet sent.

RECEIVER IN GBN:

➢ Here the receiver side window, always has **a SIZE=1.**

➢ The **NEXT EXPECTED SEQUENCE NO** , is shown with RED color text.

➢ The packets, **with GREEN color background** , are received

Correctly and ACK sent, for them.

ALGORITHM AND WORKING:

FIG 4: SENDER SIDE :

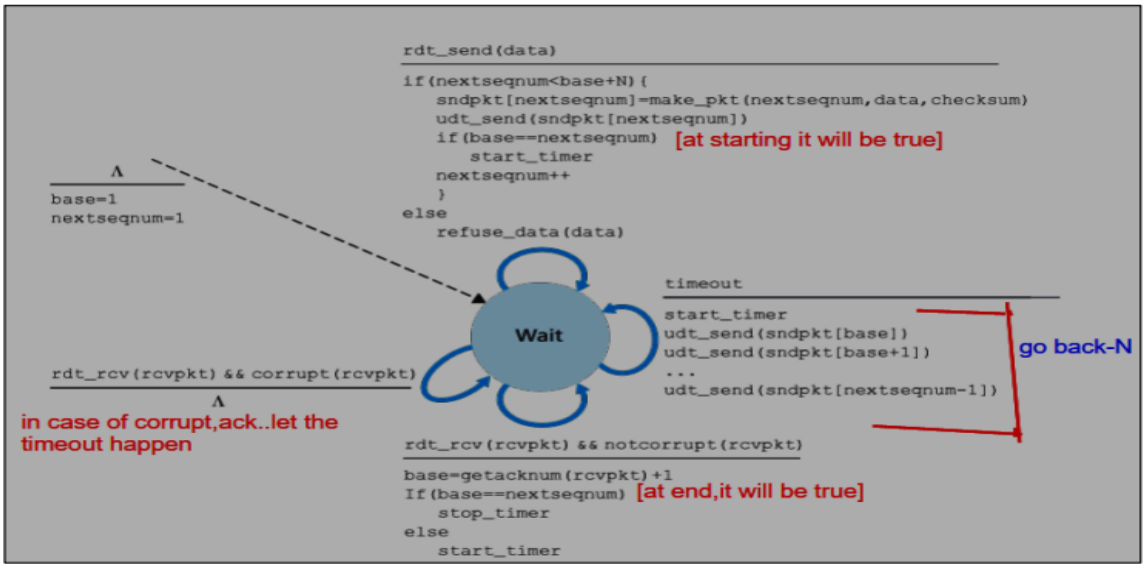
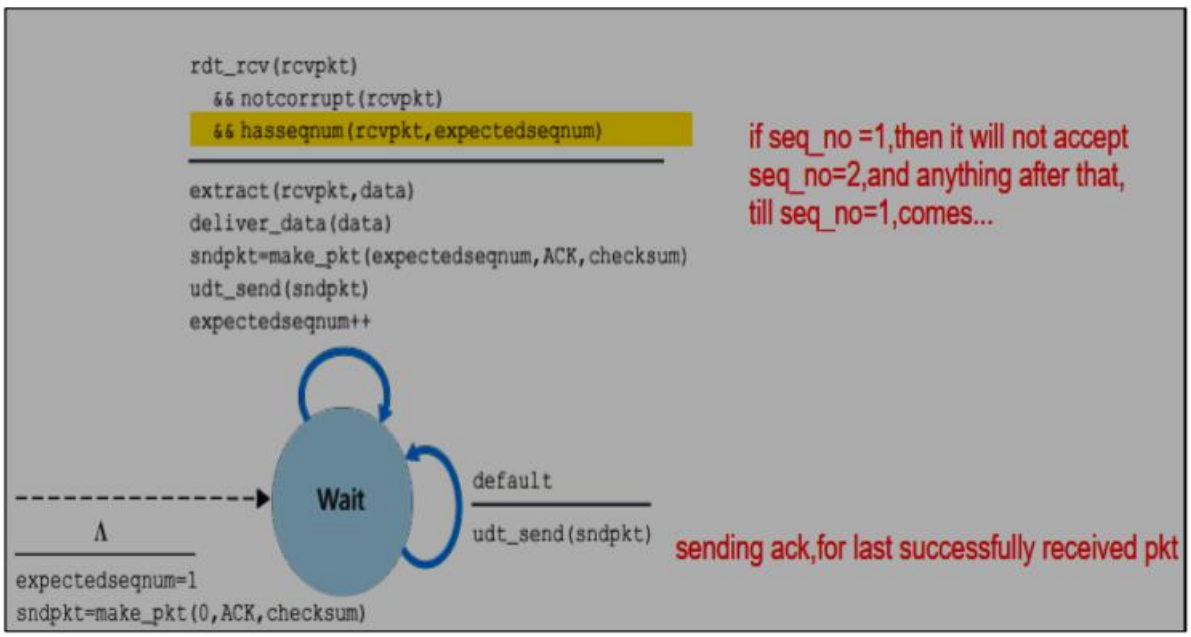


FIG 5: RECEIVER SIDE:



ALGORITHM FOR SENDER:

Algorithm 1: Go-Back-N Sender

```
Function Sender is
    send_base ← 0;
    nextseqnum ← 0;
    while True do
        if nextseqnum < send_base + N then
            send packet nextseqnum;
            nextseqnum ← nextseqnum + 1;
        end
        if receive ACK n then
            send_base ← n + 1;
            if send_base == nextseqnum then
                stop timer;
            else
                start timer;
            end
        end
        if timeout then
            start timer;
            send packet send_base;
            send packet send_base + 1;
            ...
            send packet nextseqnum − 1;
        end
    end
end
```

ALGORITHM FOR RECEVIER:

Algorithm 2: Go-Back-N Receiver

```
function Receiver is
    nextseqnum ← 0;
    while True do
        if A packet is received then
            if The received packet is not corrupted and
             sequence_number == nextseqnum then
                deliver the data to the upper layer;
                send ACK nextseqnum;
                nextseqnum ← nextseqnum + 1;
            else
                /* If the packet is corrupted or out of
                   order, simply drop it                    */
                send ACK nextseqnum − 1;
            end
        else
    end
end
```

EXPLAINATION:

➢ N is the window size of the sender, that is the MAX number of un-acknowledged packets, that a sender can send, without receiving a ACK.

➢ **“nextseqnum”** is the smallest unused SEQUENCE NUMBER, it is to be sent next, with the next packet.

➢ **“base”** is the 1st packet ,in the current window of the sender, it will be the oldest, among all the un-acknowledged packets.

➢ **Range [0,base-1] :** packets already sent, and acknowledged by receiver.

➢ **Range [base,nextseqnum-1]** are packets ,which are sent by sender, but are yet to be acknowledged ,that is, ACK is not yet received for them.

➢ **Range [nextseqnum,base+N-1]** are the sequence numbers, which can be sent immediately with a packet, as and when data is given by upper layer.

➢ **Range [base+N , till last Sequence Num]** , cannot be used, until a unacknowledged packet, in the current sender window, is acknowledged[when their ACK arrive].

➢ The reason behind having window size restrictions of N,1 in the sender, receiver is FLOW CONTROL, CONGESTION CONTROL(a important responsibility of Transport Layer)

➢ In reality, TCP has 32 bit SEQUENCE NUMBER field.

➢ max possible SEQUENCE NUMBERS in TCP: 2^{32}-1

➢ **cumulative acknowledgement:** a ACK for packet number T , indicates all packets in range [0,T] are correctly received.

➢ **When TIMEOUT happens :** when timeout happens for the **BASE** packet, then **all packets** in range [base,nextseqnum-1] are sent again simultaneously.

➢ **When a ACK is received:** when there are still packets ,which are un-acknowledged, then the TIMER is simply **restarted**. When there are no such pending packets, the TIMER is **stopped.**

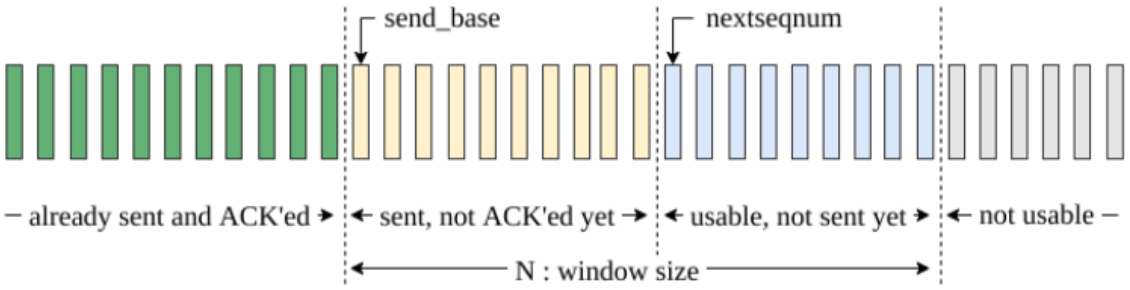
➢ In the receiver's side, only when a packet with **expectedSeqNo** is received, it is delivered to upper layer, and ACK sent. In all other cases, the received packet is discarded, and ACK of last correctly received packet ,is sent.(note: the window size of receiver side window is 1)

➢ Following must be followed to avoid ambiguity:

Sender Window Size < [number of Sequence No] -1

Receiver Window Size=1

➢ **GBN** is a example of **SLIDING WINDOW PROTOCOL**



SOURCE CODE FOR IMPLEMENTATION OF GBN PROTOCOL

RECEIVER SIDE CODE:

```
import javax.swing.*;//will act like a server(shld be executed ,before sender code)
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class receiver {
    static JButton ack,d_ack;
    static JLabel[] packett;
    static DataInputStream dis;
    static DataOutputStream dos;

    static JList<String> jl;static DefaultListModel<String> dlm;
    static JScrollBar vertical;//to controll the vertical scrollBar,of JScrollPane
    static Timer docTimer;//to autoscroll to last added entry,in JList(dlm)(we need to wait for 1 se
cond(or 700 millisec),after doing dlm.addElement(),only then the scroll,could be adjusted,so we need
 timer)

    static int expectedSeqNo=0;//next packet to be received

    receiver(){
        JFrame jf=new JFrame("Receiver in GBN");
        jf.setSize(1300,300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setLayout(new BorderLayout());
        //creating buttons
        ack=new JButton("send ack");ack.setEnabled(false);
        ack.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e5){
                try{dos.writeInt(expectedSeqNo-1);
                    dlm.addElement("ack- "+(expectedSeqNo-1)+" sent");
                }
                catch(Exception ew){
                    dlm.addElement("error:sending ack- "+(expectedSeqNo-1));
                }
                ack.setEnabled(false);d_ack.setEnabled(false);
            }
        });
        d_ack=new JButton("kill ack(default)");d_ack.setEnabled(false);
        d_ack.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e9){
                ack.setEnabled(false);
                d_ack.setEnabled(false);
                dlm.addElement("ack- "+(expectedSeqNo-1)+" killed");
            }
```

```
        });
        //creating packets
        packett=new JLabel[20];
        for(int z=0;z<20;z++){packett[z]=new JLabel(" "+z+" ");
        packett[z].setOpaque(true);
        packett[z].setBackground(Color.WHITE);}
        packett[0].setForeground(Color.red);
        //creating pane,to display summary
        dlm=new DefaultListModel<String>();
        jl=new JList<String>(dlm);
        jl.setLayoutOrientation(JList.VERTICAL);
        jl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        jl.setVisibleRowCount(20);
        JScrollPane scrollArea=new JScrollPane(jl);
        scrollArea.setSize(1000, 600);
        vertical=scrollArea.getVerticalScrollBar();
        docTimer=new Timer(700,new ActionListener(){
            public void actionPerformed(ActionEvent e19){
                vertical.validate();/*validation is,updating "Maximum" of vertical*/
                vertical.setValue( vertical.getMaximum() );}
        });
        docTimer.setRepeats(false);
        dlm.addListDataListener(new ListDataListener(){
            public void contentsChanged(ListDataEvent e99){}
            public void intervalRemoved(ListDataEvent e) {}
            public void intervalAdded(ListDataEvent e98) {
                if(docTimer.isRunning()){docTimer.restart();}
                else{docTimer.start();}}
        });


        //creating pane,to display buttons
        JPanel buttonPane=new JPanel();
        buttonPane.setLayout(new BoxLayout(buttonPane,BoxLayout.X_AXIS));
        buttonPane.add(ack);
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(d_ack);
        buttonPane.setBorder(BorderFactory.createEmptyBorder(5,360,5,120));
         // creating pane,to display packets
         JPanel packetPane=new JPanel();
         packetPane.setLayout(new FlowLayout());
         packetPane.setBorder(BorderFactory.createEmptyBorder(80, 15, 80, 15));
         for(int z=0;z<20;z++){packetPane.add(packett[z]);
            packetPane.add(Box.createHorizontalStrut(5));}
         // creating pane,for heading panel
         JPanel headingPane=new JPanel();
         JLabel heading=new JLabel("Send Window Size=1");
         headingPane.setLayout(new FlowLayout());
         headingPane.add(heading);
         //adding all panes,to main frame
        jf.add(scrollArea,BorderLayout.LINE_END);
        jf.add(buttonPane,BorderLayout.PAGE_END);
        jf.add(packetPane,BorderLayout.CENTER);
        jf.add(headingPane,BorderLayout.PAGE_START);
        jf.setVisible(true);
    }
    public static void afterReceivingPacket(){
```

```
        int w;
        try{w=dis.readInt();
            if(w==expectedSeqNo){
                dlm.addElement("packet no: "+w+" received");
                expectedSeqNo++;
                packett[expectedSeqNo-1].setForeground(Color.black);
                packett[expectedSeqNo-1].setBackground(Color.GREEN);
                if(expectedSeqNo<20){
                    packett[expectedSeqNo].setForeground(Color.red);}
            }
            else{dlm.addElement("packet no: "+w+" received,DISCARDED");}
            ack.setEnabled(true);d_ack.setEnabled(true);
        }catch(Exception ew){}
    }// func afterReceivingPacket end
    public static void resetApplication(){
        dlm.clear();dlm.addElement("Listening at port 4040");
        dlm.addElement("TCP connection estabilished!!");
        ack.setEnabled(false);
        d_ack.setEnabled(false);
        expectedSeqNo=0;
        for(int z=0;z<20;z++){packett[z].setVisible(false);
            packett[z].setBackground(Color.WHITE);
            packett[z].setForeground(Color.black);
            packett[z].setVisible(true);
        }
            packett[0].setVisible(false);
            packett[0].setForeground(Color.red);
            packett[0].setVisible(true);

    }// func resetApplication end
    public static void main(String[] args) throws Exception{
        new receiver();
        ServerSocket SS=new ServerSocket(4040);
        //System.out.println("Listening at port 4040");
        dlm.addElement("Listening at port 4040");

        Socket S=SS.accept();
        //System.out.println("Serving client..");
        dlm.addElement("TCP connection estabilished!!");

        dis=new DataInputStream(S.getInputStream());
        dos=new DataOutputStream(S.getOutputStream());

        int q;
        while(true){
            q=dis.readInt();
            switch(q){
                case 1:{S.close();SS.close();dlm.addElement("Closing Socket...");}
                case 2:{afterReceivingPacket();break;}
                case 3:{resetApplication();break;}
                case 4:{;break;}
            }//switch end

        }//while end

    }//main end
```

SENDER SIDE CODE:

```
import javax.swing.*;//will act like a client
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.time.Duration;
import java.time.LocalDateTime;
public class sender implements ActionListener{
    static JButton connectt,sendNew,killPacket,dkillPacket,resett,timerButton;
    static JLabel[] packett;

    static JList<String> jl;static DefaultListModel<String> dlm;
    static JScrollBar vertical;//to controll the vertical scrollBar,of JScrollPane
    static Timer docTimer;//to autoscroll to last added entry,in JList(dlm)(we need to wait for 1 se
cond(or 700 millisec),after doing dlm.addElement(),only then the scroll,could be adjusted,so we need
 timer)
    Socket S;
    static DataInputStream dis;
    static DataOutputStream dos;


    static Timer timerr;
    static JLabel timeDisplay;
    LocalDateTime startTime;
    Duration duration=Duration.ofSeconds(15);  // this is the duration of timer
    static int base=0;//  next packet number,to be sent
    static int nextSeqNo=0; static JLabel seqNoLabel,baseLabel;
    sender(){
        JFrame jf=new JFrame("Sender in GBN");
        jf.setSize(1300,300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setLayout(new BorderLayout());
        //creating butons
        connectt=new JButton("Create Connection");connectt.addActionListener(this);
        sendNew=new JButton("Send New");sendNew.addActionListener(this);
        sendNew.setEnabled(false);
        killPacket=new JButton("Kill Packet");killPacket.addActionListener(this);
        killPacket.setEnabled(false);
        dkillPacket=new JButton("Dont Kill Packet");dkillPacket.addActionListener(this);
        dkillPacket.setEnabled(false);
        resett=new JButton("Reset");resett.addActionListener(this);
        resett.setEnabled(false);
        timerButton=new JButton("Start Timer");
        timerButton.setEnabled(false);
        //initialising display for time,and creating pane for it
        timeDisplay=new JLabel("-- -- --");
        JLabel impPermanentInfo=new JLabel("Send Window Size=5");
```

```
        JPanel timeDisplayPanel=new JPanel();
        timeDisplayPanel.setLayout(new FlowLayout());
        timeDisplayPanel.add(impPermanentInfo);
        timeDisplayPanel.add(Box.createHorizontalStrut(120));
        timeDisplayPanel.add(timeDisplay);
        timeDisplayPanel.add(Box.createHorizontalStrut(10));
        timeDisplayPanel.add(timerButton);
        seqNoLabel=new JLabel("next Sequence no:"+nextSeqNo);
        baseLabel=new JLabel("base :"+base);
        timeDisplayPanel.add(Box.createHorizontalStrut(30));
        timeDisplayPanel.add(seqNoLabel);
        timeDisplayPanel.add(Box.createHorizontalStrut(10));
        timeDisplayPanel.add(baseLabel);
        baseLabel.setOpaque(true);
        baseLabel.setBackground(Color.white);
        seqNoLabel.setOpaque(true);
        seqNoLabel.setBackground(Color.white);
        //creating packets
        packett=new JLabel[20];
        for(int z=0;z<20;z++){
            packett[z]=new JLabel(" "+z+" ");
            packett[z].setForeground(Color.black);
            packett[z].setOpaque(true);
            packett[z].setBackground(Color.white);}
        for(int z=0;z<5;z++){
            packett[z].setForeground(Color.red);}
        //creating pane,to display summary
        dlm=new DefaultListModel<String>();
        jl=new JList<String>(dlm);jl.setVisibleRowCount(20);
        jl.setLayoutOrientation(JList.VERTICAL);
        jl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scrollArea=new JScrollPane(jl);scrollArea.setSize(1000, 300);
        vertical=scrollArea.getVerticalScrollBar();
        docTimer=new Timer(700,new ActionListener(){
            public void actionPerformed(ActionEvent e1){
                vertical.validate();/*validation is,updating "Maximum" of vertical*/
                vertical.setValue( vertical.getMaximum() );
            }
        });
        docTimer.setRepeats(false);
        dlm.addListDataListener(new ListDataListener(){
            public void contentsChanged(ListDataEvent e99){}
            public void intervalRemoved(ListDataEvent e) {}
            public void intervalAdded(ListDataEvent e98) {
                if(docTimer.isRunning()){
                    docTimer.restart();}
                else{docTimer.start();}}
        });



        //creating pane,to display buttons
        JPanel buttonPane=new JPanel();
        buttonPane.setLayout(new BoxLayout(buttonPane,BoxLayout.X_AXIS));
        buttonPane.add(connectt);
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(sendNew);
```

```
        buttonPane.add(Box.createHorizontalStrut(10));
        buttonPane.add(killPacket);
        buttonPane.add(dkillPacket);
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(resett);
        buttonPane.setBorder(BorderFactory.createEmptyBorder(5,10,5,10));
        // creating pane,to display packets
        JPanel packetPane=new JPanel();
        packetPane.setLayout(new FlowLayout());
        packetPane.setBorder(BorderFactory.createEmptyBorder(80, 15, 80, 15));
        for(int z=0;z<20;z++){
            packetPane.add(packett[z]);
            packetPane.add(Box.createHorizontalStrut(5));
        }
        //functioning timer
        timerButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e1){
                if(timerr.isRunning()){
                    timerr.stop();
                    startTime=null;
                    timerButton.setText("Start Timer");
                }
                else{
                    startTime=LocalDateTime.now();
                    timerr.start();
                    timerButton.setText("Stop Timer");
                }
            }
        });
        timerr=new Timer(300,new ActionListener(){
            public void actionPerformed(ActionEvent e3){
                LocalDateTime now=LocalDateTime.now();
                Duration runningTime=Duration.between(startTime,now);
                Duration timeLeft=duration.minus(runningTime);
                if(timeLeft.isNegative() || timeLeft.isZero()){
                    timeLeft=Duration.ZERO;
                    timerButton.doClick();
                    goBackN();
                }
                timeDisplay.setText(String.format("00h 00m %02ds", timeLeft.toSeconds()));
            }
        });
        //adding all panes,to main frame
        jf.add(scrollArea,BorderLayout.LINE_END);
        jf.add(buttonPane,BorderLayout.PAGE_END);
        jf.add(packetPane,BorderLayout.CENTER);
        jf.add(timeDisplayPanel,BorderLayout.PAGE_START);
        jf.setVisible(true);


    }//constructor end
    public static void goBackN(){
        dlm.addElement("TIMEOUT for packet no: "+base);
        dlm.addElement("Go-Back-N packets: "+base+"-"+(nextSeqNo-1));
        sendNew.setVisible(false);killPacket.setEnabled(true);
        dkillPacket.setEnabled(true);
        killPacket.setText("Kill Packets["+base+" -"+(nextSeqNo-1)+" ]");
```

```
        dkillPacket.setText("don't Kill Packets["+base+" -"+(nextSeqNo-1)+" ]");
        killPacket.setVisible(false);killPacket.setVisible(true);
        dkillPacket.setVisible(false);dkillPacket.setVisible(true);
    }
    public static void implementingGoBackN(boolean pktsKilled){
        timerButton.doClick();
        dlm.addElement("Restarting Timer for packet no:"+base);
        if(pktsKilled==false){
            try{
                for(int g=base;g<nextSeqNo;g++){
                dos.writeInt(2);
                dos.writeInt(g);
                }
                dlm.addElement("packet no: "+base+"-"+(nextSeqNo-1)+" sent");
            }catch(Exception e9){}
        }
        else{
            dlm.addElement("packet no: "+base+"-"+(nextSeqNo-1)+" got killed in network");
        }
        killPacket.setVisible(false);dkillPacket.setVisible(false);
        killPacket.setText("Kill Packet");
        dkillPacket.setText("Dont Kill Packet");
        dkillPacket.setVisible(true);killPacket.setVisible(true);

    }
    public static void sendNewPressed(boolean pktKilled){
        if(nextSeqNo<base+5){
            if(pktKilled==false){
                try{
                    dos.writeInt(2);
                    dos.writeInt(nextSeqNo);
                    dlm.addElement("packet no: "+(nextSeqNo)+" sent");}
                catch(Exception e9){
                    dlm.addElement("error sending packet no"+(nextSeqNo)+" .");}
                }
                else{
                    dlm.addElement("packet no"+(nextSeqNo)+" got killed in network");
                }

            if(base==nextSeqNo){
                if(!timerButton.isEnabled()){timerButton.setEnabled(true);}
                timerButton.doClick();
                dlm.addElement("timer started for packet no: "+(nextSeqNo)+" .");
                /*start timmer */
            }
            packett[nextSeqNo].setVisible(false);
            packett[nextSeqNo].setBackground(Color.cyan);
            packett[nextSeqNo].setVisible(true);
            nextSeqNo++;
            seqNoLabel.setText("next Sequence no:"+nextSeqNo);
            if(nextSeqNo==base+5){sendNew.setEnabled(false);}
        }
        else{dlm.addElement("sending request REJECTED-exceeding window size(5)");
         }

    }// func sendNewPressed end
    public void resetApplication(){
```

```
        base=0;nextSeqNo=0;baseLabel.setText("base :"+base);
        seqNoLabel.setText("next Sequence no:"+nextSeqNo);
        dlm.clear();if(timerr.isRunning()){timerButton.doClick();}
        timerButton.setEnabled(false);
        dlm.addElement("tcp handshaking successful");
        killPacket.setVisible(false);dkillPacket.setVisible(false);
        killPacket.setText("Kill Packet");
        dkillPacket.setText("Dont Kill Packet");
        killPacket.setEnabled(false);
        dkillPacket.setEnabled(false);killPacket.setVisible(true);
        dkillPacket.setVisible(true);
        timeDisplay.setText("-- -- --");
        sendNew.setEnabled(true);sendNew.setVisible(true);
        for(int z=0;z<20;z++){
            packett[z].setVisible(false);
            packett[z].setForeground(Color.black);
            packett[z].setBackground(Color.white);
            packett[z].setVisible(true);}
        for(int z=0;z<5;z++){
            packett[z].setVisible(false);
            packett[z].setForeground(Color.red);
            packett[z].setVisible(true);}
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==connectt){
            if(e.getActionCommand().equals("Create Connection")){
               try{ S=new Socket("localhost",4040);
                dis=new DataInputStream(S.getInputStream());
                dos=new DataOutputStream(S.getOutputStream());
                dlm.addElement("tcp handshaking successful");
                connectt.setText("Close Connection");
                sendNew.setEnabled(true);resett.setEnabled(true);
                //killPacket.setEnabled(true);dkillPacket.setEnabled(true);
                }//try end
                catch(Exception ee){
                dlm.addElement("tcp handshaking failed..");}
            }
            else{
                try{dos.writeInt(1);}
                catch(Exception eee){
                    dlm.addElement("error while closing connection.");}
                sendNew.setEnabled(false);resett.setEnabled(false);
                if(timerr.isRunning()){timerButton.doClick();};
                timerButton.setEnabled(false);
                killPacket.setEnabled(false);
                dkillPacket.setEnabled(false);
                dlm.addElement("Closing Socket...");
                connectt.setText("rerun,server & client code");
                connectt.setEnabled(false);
            }
        }//if end
        else if(e.getSource()==sendNew){
            sendNew.setVisible(false);
            killPacket.setEnabled(true);
            dkillPacket.setEnabled(true);
        }
        else if(e.getSource()==killPacket){
```

```
            if(killPacket.getText().equals("Kill Packet")){
                sendNewPressed(true);}
            else{implementingGoBackN(true);}
            sendNew.setVisible(true);
            killPacket.setEnabled(false);
            dkillPacket.setEnabled(false);
        }
        else if(e.getSource()==dkillPacket){
            if(dkillPacket.getText().equals("Dont Kill Packet")){
                sendNewPressed(false);}
            else{implementingGoBackN(false);}
            sendNew.setVisible(true);
            killPacket.setEnabled(false);
            dkillPacket.setEnabled(false);
        }
        else if(e.getSource()==resett){
            try{dos.writeInt(3);}
            catch(Exception e3){}
            resetApplication();
        }

    }//func actionPerformed end
    public static void updateBase(int b_old,int b_new){
        int y;
        for( y=b_new;(y<b_new+5)&&(y<20);y++){
            packett[y].setForeground(Color.red);}
        for(y=b_old;y<b_new;y++){
            packett[y].setVisible(false);
            packett[y].setBackground(Color.yellow);
            packett[y].setForeground(Color.black);
            packett[y].setVisible(true);
        /*setVisible() from false to true,will repaint the component*/
        }
        if(!sendNew.isEnabled() &&(nextSeqNo-b_new<5)){
            sendNew.setEnabled(true);
        }// as only 5 packets,are allowed to be sent,without receiving ack,so sendnew button was dis
abled
        baseLabel.setText("base :"+b_new);
    }

    public static void main(String[] args)throws Exception {
        new sender();
        //receving ack
        int w=0;
        while(true){
            try{w=dis.readInt();
                if(killPacket.getText().equals("Kill Packet")){
                    // to ensure implementingGoBackN() is not in progress
                    dlm.addElement("ack: "+w+" received.");
                    updateBase(base,w+1);
                    base=w+1;
                    if(base==nextSeqNo){
                        timerButton.doClick();
                    timerButton.setEnabled(false);
                    dlm.addElement("Stoping Timer");/*stop timmer */
                    }
                    else{
```

```
                    timerButton.doClick();
                    timerButton.doClick();
                    dlm.addElement("Restarting Timer");
                    /*restart timer */
                }
            }
            else{dlm.addElement("ack: "+w+" discarded, as GBN procedure is in progress");
        /*ACK received while implementing func: implementingGoBackN() will not be considered */}

        }
        catch(Exception e){
            /*dlm.addElement("error:while receving ack");error coming,*/}


    }
  }//main end
}//class end
```

SELECTIVE REPEAT PROTOCOL (SR PROTOCOL)

DISADVANTAGES OF GBN:

• A single error, in the 1st packet, will make the rest of the successive packets ,transmitted as waste. And will lead to **RE-TRANSMISSION**, which obviously can be avoided.

• When window size is large, and there is high congestion in network, the probability of 1st packet of a retransmission getting dropped ,by a intermediate router is **high.** This adds to ,number of retransmissions, in a **already congested channel.**

• The window size of 1 in receiver side, and large window size, on sender side, perform extremely bad, in a congested network (where changes of a packet, getting dropped is high).

• It does not utilize the benefit of **buffers**, that can be added to both hosts, which will **reduce**, the **discarding of,rightly** received packets(which were received out of sequence).

• With buffers being introduced, the receiver, no longer needs to limit its window size to 1.

• ALL THESE IDEAS, ARE INCOPERATED IN SR PROTOCOL.

DISADVANTAGES OF SR:

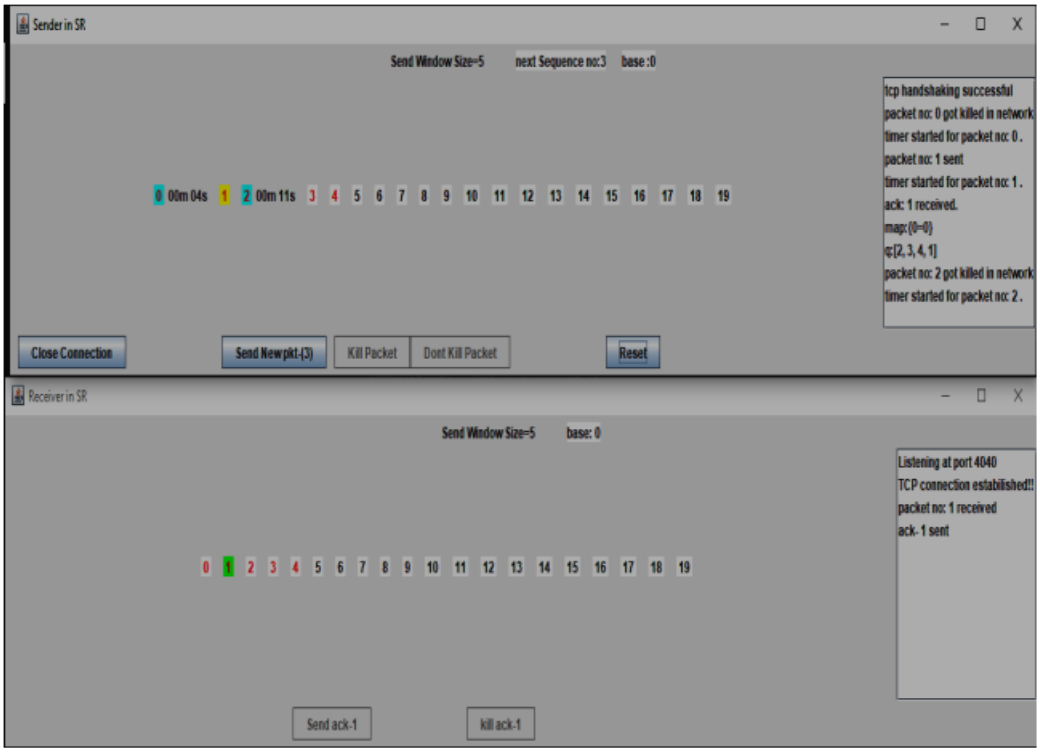
• **Hardware limitations,** play a big role in deciding the

window size, as each sequence number, within the senderside window**, needs to be allotted a timer**, which must be capable of running simultaneously.(all N timers, running simultaneously).

• **Hardware limitations,** in another form, will be the **amount of Buffer, available especially for the RECEIVER side,** this factor will limit the window size.

• **Congestion** in the channel, is another factor, based on which , **window size, should be determined**. But Congestion changes with time, thus is **un-predictable.**

EXPLANATION ABOUT ITS WORKING:



Sender in SR Protocol:

➢ In this case N=5 ,is set as Default. That is the pipeline can have a Max of 5 unacknowledged packet ,at any time.

➢ Here we have taken **SEQUENCE NUMBER** of range [0-19],for simplicity, of GUI.

➢ The packets currently lying inside the sender side window of size=5, are printed with **RED** ink. Example : [0,1,2,3,4] in the above picture

➢ The packets with a **YELLOW** background color are acknowledged, by receiver.

➢ The packets with **BLUE** background color, are sent, but the sender is yet to receive ACK for them. Each such packet, has a corresponding TIMER to its right.

➢ Here **the MAXIMUM WAITING time**, for a packet is taken as 15 seconds, for the purpose of simplicity.(maximum waiting time, must be greater than :- **[RTT + processing time by receiver]** , in reality.)

➢ The **"base", "next Sequence No"** field, are updated with each new packet sent, ACK received.

➢ The packets with **WHITE** are, not yet sent.

➢ Each packet within the Sender side Window, is allotted a **TIMER.** When a packet is sent, its timer is **started.** When a ACK is received for a sent packet, its timer is **stopped.**

➢ When timeout happens for a packet, that particular packet is sent again, and timer is restarted. Thus the name SELECTIVE REPEAT.

RECEIVER IN SR:

➢ Here the receiver side window, always has a size same as Sender side, that is **SIZE=5=N.**

In this case it is [0,1,2,3,4].

➢ The **packets within window** , is shown with **RED** color text.

➢ The packets, **with GREEN color background** , are received correctly and ACK sent, for them.

➢ For receiver side window also, there exists a unique **"base".**

ALGORITHM AND WORKING:

Sender Side Algorithm:

1. When the upper layer calls for sending data, it is checked ,If a unused sequence number is available, within window , if it is available, the data is made into a packet, required headers are added and sent. Otherwise it is buffered.

2. TIMEOUT EVENT: it is designed to protect against overly delayed/ lost packets .Unlike GBN, each packet in SR has a logical timer, and timeout of a particular timer, results

in RESENDING the packet, corresponding to that timer. The timer is started/stopped/restarted based upon packet getting sent/ACK received/resent .

3. ACK RECEIVED: if the ACK is for the base packet of window, the window is then slided, such that the new base now is the nearest unacknowledged packet, to the previous base. New sequence numbers, which are now inside window, now can be utilized to send a packet, on demand, instantaneously.

Receiver Side Algorithm:

1. Any packet with sequence number in Range: [base,base+N-1] are correctly received .And a ACK is sent for that sequence number. If the packet is in the above range, but not in correct order, than ,they need to be buffered. When a packet ,with sequence number=base of receiver, arrives, then this packet, and previously buffered consecutive packets are together delivered, to its upper layer.(and window is slided, similar to sender side case).

2. For packets with sequence number in range : [0, base-1] ,a DUPLICATE ACK (as receiver has already acknowledged this packet) is sent.

3. In all other cases, received packet is discarded.

EXPLAINATION:

- It works similar to GBN protocol, except that each packet in sender side, has a unique logical timer, and receiver side window is same to size of sender side.

- Receiver side, **BUFFERS** the packet received out of sequence, and sends them to upper layer, only when all the packets, in sequence are received, in whatever order.

- This drastically reduces, the number of packets discarded by receiver.

- It is very efficient, even in case of a congested channel, as the sender always sends only 1 packet, (even during re-transmission).

- The number of correctly received packets, being discarded is also low, as its window size is same as sender side, which really helps in a contested channel .

- Following must be followed to avoid ambiguity:

Sender Window Size < =([number of Sequence No]/2) -1

Receiver Window Size=Sender window size=N

SOURCE CODE FOR IMPLEMENTATION OF SR PROTOCOL

RECEIVER SIDE CODE :

```
import javax.swing.*;//will act like a server(shld be executed ,before sender code)
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class receiver {
    static JButton ack,d_ack;
    static JLabel[] packett;
    static DataInputStream dis;
    static DataOutputStream dos;

    static JList<String> jl;static DefaultListModel<String> dlm;
    static JScrollBar vertical;//to controll the vertical scrollBar,of JScrollPane
    static Timer docTimer;//to autoscroll to last added entry,in JList(dlm)(we need to wait for 1 se
cond(or 700 millisec),after doing dlm.addElement(),only then the scroll,could be adjusted,so we need
 timer)

    static int base=0;static JLabel baseLabel;
    static boolean duplicateAck=false;
    static int w=-1;//received pkt no
    //static int nextSeqNo=0;//next packet to be received

    receiver(){
        JFrame jf=new JFrame("Receiver in SR");
        jf.setSize(1300,300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setLayout(new BorderLayout());
        //creating buttons
        ack=new JButton("send ack");
        ack.setEnabled(false);
        ack.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e5){
                try{dos.writeInt(w);
                    if(duplicateAck){
                        dlm.addElement("Duplicate ack- "+(w)+" sent");
                        duplicateAck=false;}
                    else{
                        dlm.addElement("ack- "+(w)+" sent");}
                    }
                catch(Exception ew){
                    dlm.addElement("error:sending ack- "+w);}
                ack.setEnabled(false);
                d_ack.setEnabled(false);
            }
```

```
        });
        d_ack=new JButton("kill ack");
        d_ack.setEnabled(false);
        d_ack.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e9){
                ack.setEnabled(false);
                d_ack.setEnabled(false);
                if(duplicateAck){
                    dlm.addElement("Duplicate ack- "+(w)+" killed");}
                else{dlm.addElement("ack- "+(w)+" killed");}
                duplicateAck=false;
            }
        });
        //creating packets
        packett=new JLabel[20];
        for(int z=0;z<20;z++){
            packett[z]=new JLabel(" "+z+" ");
            packett[z].setOpaque(true);
            packett[z].setBackground(Color.WHITE);}
        for(int z=0;z<5;z++){
            packett[z].setForeground(Color.red);}
        //creating pane,to display summary
        dlm=new DefaultListModel<String>();
        jl=new JList<String>(dlm);
        jl.setLayoutOrientation(JList.VERTICAL);
        jl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        jl.setVisibleRowCount(20);
        JScrollPane scrollArea=new JScrollPane(jl);
        scrollArea.setSize(1000, 600);
        vertical=scrollArea.getVerticalScrollBar();
        docTimer=new Timer(700,new ActionListener(){
            public void actionPerformed(ActionEvent e19){
                vertical.validate();/*validation is,updating "Maximum" of vertical*/
                vertical.setValue( vertical.getMaximum() );}
        });
        docTimer.setRepeats(false);
        dlm.addListDataListener(new ListDataListener(){
            public void contentsChanged(ListDataEvent e99){}
            public void intervalRemoved(ListDataEvent e) {}
            public void intervalAdded(ListDataEvent e98) {
                if(docTimer.isRunning()){
                    docTimer.restart();}
                else{docTimer.start();}}
        });


        //creating pane,to display buttons
        JPanel buttonPane=new JPanel();
        buttonPane.setLayout(new BoxLayout(buttonPane,BoxLayout.X_AXIS));
        buttonPane.add(ack);
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(d_ack);
        buttonPane.setBorder(BorderFactory.createEmptyBorder(5,360,5,120));
         // creating pane,to display packets
         JPanel packetPane=new JPanel();
         packetPane.setLayout(new FlowLayout());
         packetPane.setBorder(BorderFactory.createEmptyBorder(80, 15, 80, 15));
```

```
        for(int z=0;z<20;z++){
            packetPane.add(packett[z]);
            packetPane.add(Box.createHorizontalStrut(5));}
        // creating pane,for heading panel
        JPanel headingPane=new JPanel();
        JLabel heading=new JLabel("Send Window Size=5");
        baseLabel=new JLabel("base: "+0);
        baseLabel.setOpaque(true);
        baseLabel.setBackground(Color.WHITE);
        headingPane.setLayout(new FlowLayout());
        headingPane.add(heading);
        headingPane.add(Box.createHorizontalStrut(30));
        headingPane.add(baseLabel);

        //adding all panes,to main frame
       jf.add(scrollArea,BorderLayout.LINE_END);
       jf.add(buttonPane,BorderLayout.PAGE_END);
       jf.add(packetPane,BorderLayout.CENTER);
       jf.add(headingPane,BorderLayout.PAGE_START);
       jf.setVisible(true);
    }// constructor end
    public static void updateBase(){
        int b_old=base;int y;
        for( y=b_old;((y<b_old+5)&&(y<20));y++){
            if(packett[y].getBackground()==Color.GREEN){base=y+1;}
            else{break;}
        }
        dlm.addElement("b_old:"+b_old+" b_new:"+base);
        for(y=b_old;y<base;y++){
            packett[y].setVisible(false);
            packett[y].setForeground(Color.BLACK);
            packett[y].setVisible(true);}
        for(y=base;((y<base+5)&&(y<20));y++){
            packett[y].setVisible(false);
            packett[y].setForeground(Color.RED);
            packett[y].setVisible(true);}
        baseLabel.setText("base: "+base);
    }//func updateBase end
    public static void afterReceivingPacket(){
        try{w=dis.readInt();// w is received pkt no
            if((w>=base)&&(w<base+5)){
                dlm.addElement("packet no: "+w+" received");
                packett[w].setBackground(Color.GREEN);
                if(w==base){updateBase();}
            }
            else{
                dlm.addElement("packet no: "+w+" received,AGAIN");
                duplicateAck=true;
            }
            ack.setVisible(false);d_ack.setVisible(false);
            if(duplicateAck){
                ack.setText("Send Duplicate ack-"+w);
                d_ack.setText("Kill Duplicate ack-"+w);}
            else{
                ack.setText("Send ack-"+w);
                d_ack.setText("kill ack-"+w);}
            ack.setEnabled(true);d_ack.setEnabled(true);
```

```
            ack.setVisible(true);d_ack.setVisible(true);
        }catch(Exception ew){}
    }// func afterReceivingPacket end
    public static void resetApplication(){
        dlm.clear();
        dlm.addElement("Listening at port 4040");
        dlm.addElement("TCP connection estabilished!!");
        ack.setEnabled(false);d_ack.setEnabled(false);
        base=0;duplicateAck=false;
        w=-1;
        baseLabel.setText("base: "+base);
        for(int z=0;z<20;z++){
            packett[z].setVisible(false);
            packett[z].setBackground(Color.WHITE);
            packett[z].setForeground(Color.black);
            packett[z].setVisible(true);}
        for(int z=0;z<5;z++){
            packett[z].setVisible(false);
            packett[z].setForeground(Color.red);
            packett[z].setVisible(true);}

    }// func resetApplication end
    public static void main(String[] args) throws Exception{
        new receiver();
        ServerSocket SS=new ServerSocket(4040);
        //System.out.println("Listening at port 4040");
        dlm.addElement("Listening at port 4040");

        Socket S=SS.accept();
        //System.out.println("Serving client..");
        dlm.addElement("TCP connection estabilished!!");

        dis=new DataInputStream(S.getInputStream());
        dos=new DataOutputStream(S.getOutputStream());

        int q;
        while(true){
            q=dis.readInt();
            switch(q){
                case 1:{S.close();SS.close();dlm.addElement("Closing Socket...");}
                case 2:{afterReceivingPacket();break;}
                case 3:{resetApplication();break;}
                case 4:{;break;}
            }//switch end

        }//while end

    }//main end

}//class end
```

SENDER SIDE CODE:

```
import javax.swing.*;//will act like a client
import javax.swing.event.ListDataEvent;
import javax.swing.event.ListDataListener;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;

public class sender implements ActionListener{
    static JButton connectt,sendNew,killPacket,dkillPacket,resett;
    static JLabel[] packett;

    static JList<String> jl;static DefaultListModel<String> dlm;
    static JScrollBar vertical;//to controll the vertical scrollBar,of JScrollPane
    static Timer docTimer;//To autoscroll to last added entry,in JList(dlm)(we need to wait for 1 se
cond(or 700 millisec),after doing dlm.addElement(),only then the scroll,could be adjusted,so we need
 timer)
    Socket S;
    static DataInputStream dis;
    static DataOutputStream dos;


    static Timer[] timerr;
    static JLabel[] timeDisplay;
    static Queue<Integer> q=new LinkedList<>();// to store unused timers
    static Queue<Integer> t=new LinkedList<>();// to store pktNo,for which timer TIMED-OUT
    static HashMap<Integer,Integer> map=new HashMap<Integer,Integer>();// to map sequenceNo of pkt,w
ith its allocated timmer index
    static LocalDateTime[] startTime;
    static  Duration duration=Duration.ofSeconds(15);  // this is the duration of timer
    static int base=0;//  next packet number,to be sent
    static int nextSeqNo=0; static JLabel seqNoLabel,baseLabel;
    sender(){
        JFrame jf=new JFrame("Sender in SR");
        jf.setSize(1300,300);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setLayout(new BorderLayout());
        //creating butons
        connectt=new JButton("Create Connection");connectt.addActionListener(this);
        sendNew=new JButton("Send New pkt-(0)");sendNew.addActionListener(this);
        sendNew.setEnabled(false);
        killPacket=new JButton("Kill Packet");killPacket.addActionListener(this);
        killPacket.setEnabled(false);
        dkillPacket=new JButton("Dont Kill Packet");dkillPacket.addActionListener(this);
        dkillPacket.setEnabled(false);
        resett=new JButton("Reset");resett.addActionListener(this);
        resett.setEnabled(false);
        //initialising display for time,and creating pane for it
        timeDisplay=new JLabel[20];
```

```
        for(int z=0;z<20;z++){
            timeDisplay[z]=new JLabel("00m 00s");
            timeDisplay[z].setVisible(false);}
        JLabel impPermanentInfo=new JLabel("Send Window Size=5");
        JPanel timeDisplayPanel=new JPanel();
        timeDisplayPanel.setLayout(new FlowLayout());
        timeDisplayPanel.add(impPermanentInfo);
        seqNoLabel=new JLabel("next Sequence no:"+nextSeqNo);
        baseLabel=new JLabel("base :"+base);
        timeDisplayPanel.add(Box.createHorizontalStrut(30));
        timeDisplayPanel.add(seqNoLabel);
        timeDisplayPanel.add(Box.createHorizontalStrut(10));
        timeDisplayPanel.add(baseLabel);
        baseLabel.setOpaque(true);
        baseLabel.setBackground(Color.white);
        seqNoLabel.setOpaque(true);seqNoLabel.setBackground(Color.white);
        //creating packets
        packett=new JLabel[20];
        for(int z=0;z<20;z++){
            packett[z]=new JLabel(" "+z+" ");
            packett[z].setForeground(Color.black);
            packett[z].setOpaque(true);
            packett[z].setBackground(Color.white);}
        for(int z=0;z<5;z++){
            packett[z].setForeground(Color.red);}

        //creating pane,to display summary
        dlm=new DefaultListModel<String>();
        jl=new JList<String>(dlm);
        jl.setVisibleRowCount(20);jl.setLayoutOrientation(JList.VERTICAL);
        jl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scrollArea=new JScrollPane(jl);scrollArea.setSize(1000, 300);
        vertical=scrollArea.getVerticalScrollBar();
        docTimer=new Timer(700,new ActionListener(){
            public void actionPerformed(ActionEvent e1){
                vertical.validate();/*validation is,updating "Maximum" of vertical*/
                vertical.setValue( vertical.getMaximum() );}
        });
        docTimer.setRepeats(false);
        dlm.addListDataListener(new ListDataListener(){
            public void contentsChanged(ListDataEvent e99){}
            public void intervalRemoved(ListDataEvent e) {}
            public void intervalAdded(ListDataEvent e98) {
                if(docTimer.isRunning()){docTimer.restart();}
                else{docTimer.start();}}
        });



        //creating pane,to display buttons
        JPanel buttonPane=new JPanel();
        buttonPane.setLayout(new BoxLayout(buttonPane,BoxLayout.X_AXIS));
        buttonPane.add(connectt);
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(sendNew);
        buttonPane.add(Box.createHorizontalStrut(10));
        buttonPane.add(killPacket);buttonPane.add(dkillPacket);
```

```
        buttonPane.add(Box.createHorizontalStrut(120));
        buttonPane.add(resett);
        buttonPane.setBorder(BorderFactory.createEmptyBorder(5,10,5,10));
        // creating pane,to display packets,their corresponding timers
        JPanel packetPane=new JPanel();
        packetPane.setLayout(new FlowLayout());
        packetPane.setBorder(BorderFactory.createEmptyBorder(80, 15, 80, 15));
        for(int z=0;z<20;z++){
            packetPane.add(packett[z]);
            packetPane.add(timeDisplay[z]);
            packetPane.add(Box.createHorizontalStrut(5));}

        //initialising timer,and other related variables

        startTime=new LocalDateTime[5];

        timerr=new Timer[5];


        //adding all panes,to main frame
        jf.add(scrollArea,BorderLayout.LINE_END);
        jf.add(buttonPane,BorderLayout.PAGE_END);
        jf.add(packetPane,BorderLayout.CENTER);
        jf.add(timeDisplayPanel,BorderLayout.PAGE_START);
        jf.setVisible(true);


    }//constructor end
    public static void startTimerFor(int pkttNo){
        if(q.size()>0){
            int timerIndexLeft=q.poll();
            timerr[timerIndexLeft]=new Timer(300,new ActionListener(){
                public void actionPerformed(ActionEvent e3){
                    LocalDateTime now=LocalDateTime.now();//System.out.println("1");
                    Duration runningTime=Duration.between(startTime[timerIndexLeft],now);//System.ou
t.println("2");
                    Duration timeLeft=duration.minus(runningTime);//System.out.println("3");
                    if(timeLeft.isNegative() || timeLeft.isZero()){
                        timeLeft=Duration.ZERO;
                        timerr[timerIndexLeft].stop();
                        selectiveRepeat(pkttNo);/*TIMEOUT HAPPENED*/}
                    timeDisplay[pkttNo].setText(String.format("00m %02ds", timeLeft.toSeconds()));//
System.out.println("4");
                }
            });
            startTime[timerIndexLeft]=LocalDateTime.now();//System.out.println("5");
            timerr[timerIndexLeft].start();//System.out.println("6");
            map.put(pkttNo,timerIndexLeft);//dlm.addElement("map:"+map.toString());dlm.addElement("q
:"+q);
        }else{dlm.addElement("error:no timer to begin!!");}

    }
    public static void selectiveRepeat(int pkktNo){
        dlm.addElement("TIMEOUT for packet no: "+pkktNo);
        if(killPacket.isEnabled()){

        }// we cant edit the text of killPacket,at this moment
```

```
        else{
            sendNew.setVisible(false);/*sendNew is not visible,now*/
            killPacket.setEnabled(true);dkillPacket.setEnabled(true);
            killPacket.setVisible(false);dkillPacket.setVisible(false);
            killPacket.setText("Kill ReSent Packet-"+pkktNo);
            dkillPacket.setText("Dont Kill ReSent Packet-"+pkktNo);
            killPacket.setVisible(true);dkillPacket.setVisible(true);}

        t.add(pkktNo);
    }
    public static void implementingSelectiveRepeat(boolean pktsKilled){
        int pkktNo=t.poll();

        startTime[map.get(pkktNo)]=LocalDateTime.now();/*vv imp*/
        timerr[map.get(pkktNo)].restart();
        dlm.addElement("Restarting Timer for packet no:"+pkktNo);/*restarting timed-out timer */

        if(pktsKilled==false){
            try{
                dos.writeInt(2);dos.writeInt(pkktNo);
                dlm.addElement("packet no: "+pkktNo+" Re-sent");
            }catch(Exception e9){}
        }
        else{dlm.addElement("packet no: "+pkktNo+" got killed in network");}
        if(t.size()>0){
            pkktNo=t.peek();
            killPacket.setVisible(false);dkillPacket.setVisible(false);
            killPacket.setText("Kill ReSent Packet-"+pkktNo);
            dkillPacket.setText("Dont Kill ReSent Packet-"+pkktNo);
            killPacket.setVisible(true);dkillPacket.setVisible(true);
            return;}
        killPacket.setVisible(false);dkillPacket.setVisible(false);
        killPacket.setText("Kill Packet");dkillPacket.setText("Dont Kill Packet");
        dkillPacket.setVisible(true);killPacket.setVisible(true);
        sendNew.setVisible(true);killPacket.setEnabled(false);
        dkillPacket.setEnabled(false);

    }
    public static boolean sendNewPressed(boolean pktKilled){
        if(nextSeqNo<base+5){
            if(pktKilled==false){
                try{
                    dos.writeInt(2);
                    dos.writeInt(nextSeqNo);
                    dlm.addElement("packet no: "+(nextSeqNo)+" sent");}//try end
                catch(Exception e9){
                    dlm.addElement("error sending packet no: "+(nextSeqNo)+" .");
                    dlm.addElement("CHECK CONNECTION,TO RECEIVER!!!");}/*catch end */
            }
            else{
                dlm.addElement("packet no: "+(nextSeqNo)+" got killed in network");}

            timeDisplay[nextSeqNo].setVisible(true);/*enabling JLabel to show timer readings,for thi
s packet */
            startTimerFor(nextSeqNo);
            dlm.addElement("timer started for packet no: "+(nextSeqNo)+" .");/*start timmer */
            packett[nextSeqNo].setVisible(false);
```

```
            packett[nextSeqNo].setBackground(Color.cyan);
            packett[nextSeqNo].setVisible(true);
            nextSeqNo++;seqNoLabel.setText("next Sequence no:"+nextSeqNo);
            if((nextSeqNo==base+5)||(nextSeqNo>=20)){
                sendNew.setEnabled(false);}
            sendNew.setText("Send New pkt-("+nextSeqNo+")");
        }
        else{dlm.addElement("sending request REJECTED-exceeding window size(5)");}

        if(t.size()>0){
             return true;}
        else{return false;}// checking if any timer got timed out,while user was selecting (to kill)
/(dont kill) new packet
    }// func sendNewPressed end
    public void resetApplication(){
        base=0;nextSeqNo=0;
        baseLabel.setText("base :"+base);
        seqNoLabel.setText("next Sequence no:"+nextSeqNo);
        dlm.clear();dlm.addElement("tcp handshaking successful");
        killPacket.setVisible(false);dkillPacket.setVisible(false);
        killPacket.setText("Kill Packet");
        dkillPacket.setText("Dont Kill Packet");
        killPacket.setEnabled(false);dkillPacket.setEnabled(false);
        killPacket.setVisible(true);dkillPacket.setVisible(true);
        for(int z=0;z<20;z++){
            timeDisplay[z].setVisible(false);
            timeDisplay[z].setText("00m 00s");
            /*timeDisplay[z].setVisible(true);*/
        }
        sendNew.setEnabled(true);sendNew.setVisible(false);
        sendNew.setText("Send New pkt-(0)");
        sendNew.setVisible(true);
        for(int z=0;z<20;z++){
            packett[z].setVisible(false);
            packett[z].setForeground(Color.black);
            packett[z].setBackground(Color.white);
            packett[z].setVisible(true);}
        for(int z=0;z<5;z++){
            packett[z].setVisible(false);
            packett[z].setForeground(Color.red);
            packett[z].setVisible(true);}
        for(int z=0;z<5;z++){
            if((timerr[z]!=null)&&(timerr[z].isRunning())){
                timerr[z].stop();}
        }
        q.clear();
        q.add(0);q.add(1);q.add(2);q.add(3);q.add(4);
        map.clear();
        t.clear();
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==connectt){
            if(e.getActionCommand().equals("Create Connection")){

               try{ S=new Socket("localhost",4040);
                dis=new DataInputStream(S.getInputStream());
                dos=new DataOutputStream(S.getOutputStream());
```

```
            dlm.addElement("tcp handshaking successful");
            connectt.setText("Close Connection");

            sendNew.setEnabled(true);resett.setEnabled(true);
            }//try end
            catch(Exception ee){
                dlm.addElement("tcp handshaking failed..");}
        }
        else{
            try{dos.writeInt(1);}
            catch(Exception eee){
                dlm.addElement("error while closing connection.");}
            sendNew.setEnabled(false);
            resett.setEnabled(false);
            /*if(timerr.isRunning()){timerButton.doClick();};*/
            killPacket.setEnabled(false);dkillPacket.setEnabled(false);
            dlm.addElement("Closing Socket...");
            connectt.setText("rerun,server & client code");
            connectt.setEnabled(false);
        }
    }//if end
    else if(e.getSource()==sendNew){
        sendNew.setVisible(false);
        killPacket.setEnabled(true);dkillPacket.setEnabled(true);
    }
    else if(e.getSource()==killPacket){
        if(killPacket.getText().equals("Kill Packet")){
            if(sendNewPressed(true)){
                killPacket.setVisible(false);
                dkillPacket.setVisible(false);
                killPacket.setText("Kill ReSent Packet-"+t.peek());
                dkillPacket.setText("Dont Kill ReSent Packet-"+t.peek());
                killPacket.setVisible(true);dkillPacket.setVisible(true);}
            else{
                sendNew.setVisible(true);
                killPacket.setEnabled(false);dkillPacket.setEnabled(false);}
        }
        else{implementingSelectiveRepeat(true);}
    }
    else if(e.getSource()==dkillPacket){
        if(dkillPacket.getText().equals("Dont Kill Packet")){
            if(sendNewPressed(false)){
                killPacket.setVisible(false);dkillPacket.setVisible(false);
                killPacket.setText("Kill ReSent Packet-"+t.peek());
                dkillPacket.setText("Dont Kill ReSent Packet-"+t.peek());
                killPacket.setVisible(true);dkillPacket.setVisible(true);}
            else{
                sendNew.setVisible(true);
                killPacket.setEnabled(false);dkillPacket.setEnabled(false);}
            }
        else{implementingSelectiveRepeat(false);}
    }
    else if(e.getSource()==resett){
        try{dos.writeInt(3);}
        catch(Exception e3){}
        resetApplication();
    }
```

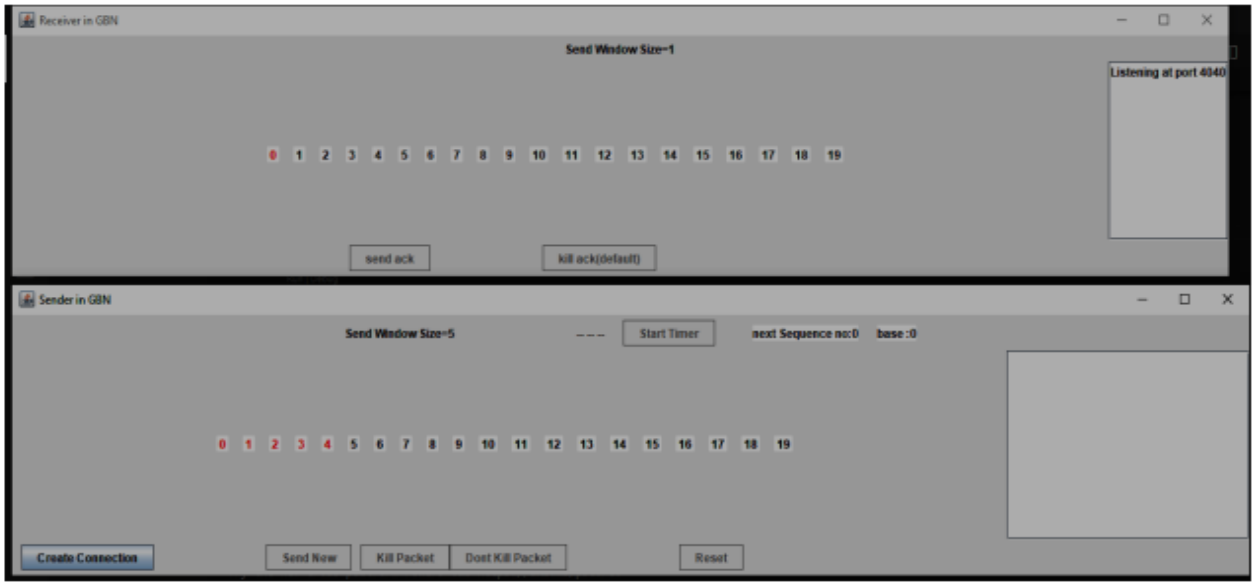
```
    }//func actionPerformed end
    public static void updateBase(){
        int y;int b_old=base;
        for(y=b_old;((y<20)&&(y<nextSeqNo));y++){
            if(!map.containsKey(y)){base=y+1;}
            else{break;}
        }// System.out.println("b_old: "+b_old+"   b_new:"+base);
        for( y=base;((y<base+5)&&(y<20));y++){
            packett[y].setForeground(Color.red);}
        for(y=b_old;y<base;y++){
            packett[y].setVisible(false);packett[y].setBackground(Color.yellow);
            packett[y].setForeground(Color.black);packett[y].setVisible(true);
            /*setVisible() from false to true,will repaint the component*/}
        if(!sendNew.isEnabled() &&(nextSeqNo-base<5)){
            sendNew.setEnabled(true);}// as only 5 packets,are allowed to be sent,without receiving
ack,so sendnew button was disabled
        if(nextSeqNo>=20){sendNew.setEnabled(false);}
        baseLabel.setText("base :"+base);
    }

    public static void main(String[] args)throws Exception {
        new sender();
        //initialising queue
        q.add(0);q.add(1);q.add(2);
        q.add(3);q.add(4);
        //dlm.addElement("map(initial):"+map.toString());dlm.addElement("q(initial):"+q);
        //receving ack
        int w=0;
        while(true){
            try{
                w=dis.readInt();

                if(t.contains(w)){
                    dlm.addElement("ack: "+w+" discarded,as selective Repeat procedure is being impl
emented for it.");}
                else{   dlm.addElement("ack: "+w+" received.");
                        if(map.containsKey(w)){
                            timerr[map.get(w)].stop();
                            q.add(map.remove(w));
                            // dlm.addElement("map:"+map.toString());
                            // dlm.addElement("q:"+q);
                            // dlm.addElement("t:"+t);
                            timeDisplay[w].setVisible(false);}
                        packett[w].setVisible(false);
                        packett[w].setBackground(Color.yellow);
                        packett[w].setVisible(true);
                        if(w==base){updateBase();}
                }

            }
            catch(Exception e){/*dlm.addElement("error:while receving ack");error coming,*/}


        }
    }//main end
}//class end
```

SNAPSHOTS OF APPLICATION

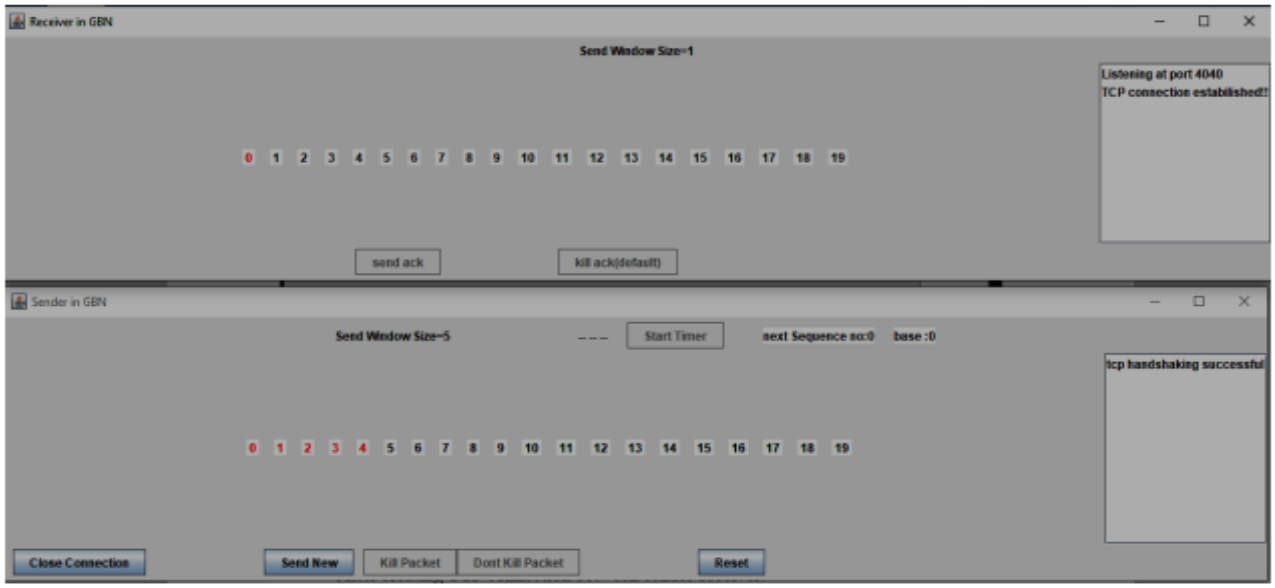
GBN APPLICATION, AT ITS VARIOUS STAGES:

Case 1:

- App at starting:

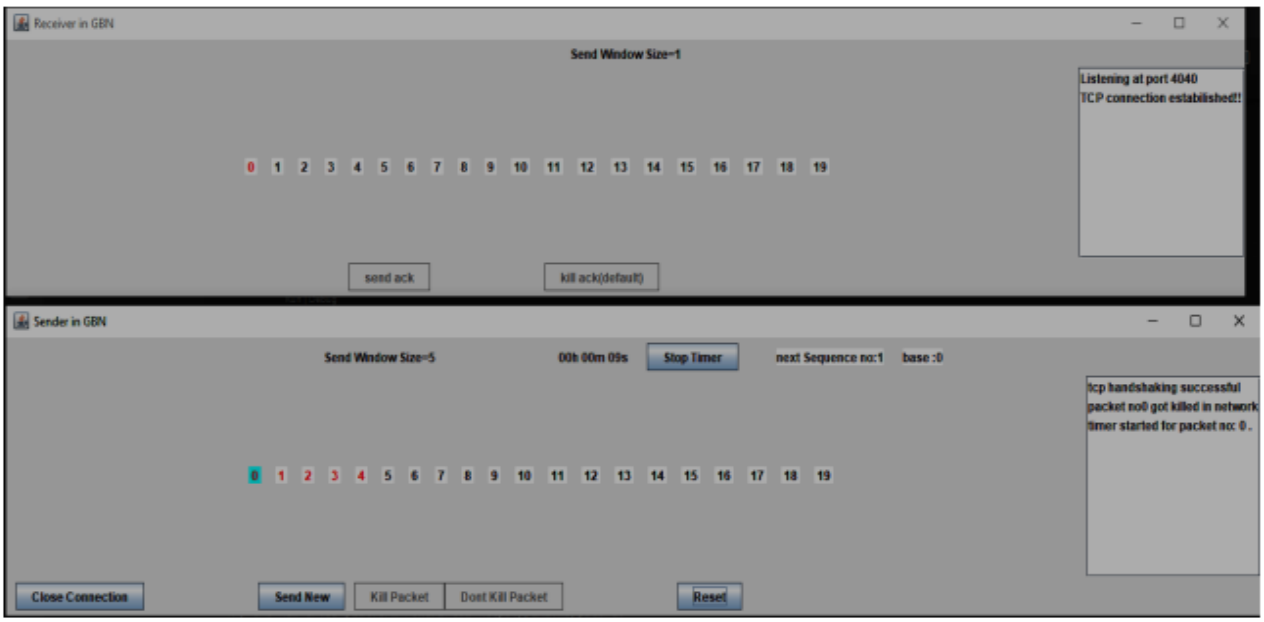


- After creating TCP connection between sender-receiver:



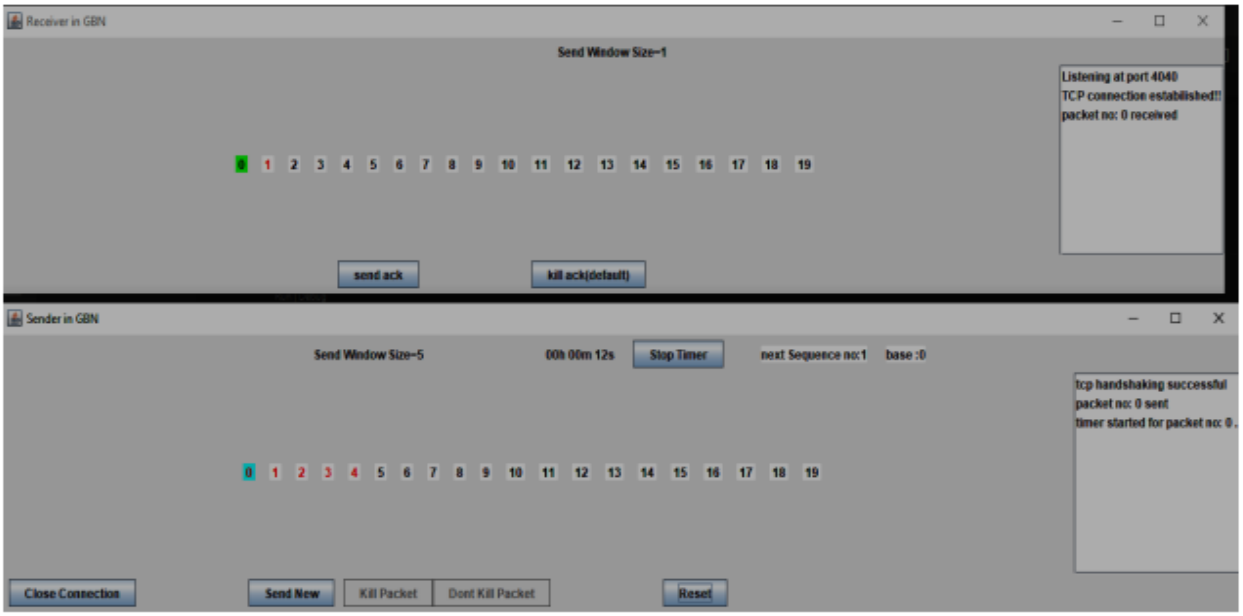
• Options we have after pressing “Send New” in Sender Side:

• When “ Kill Packet” is Pressed in Sender Side:



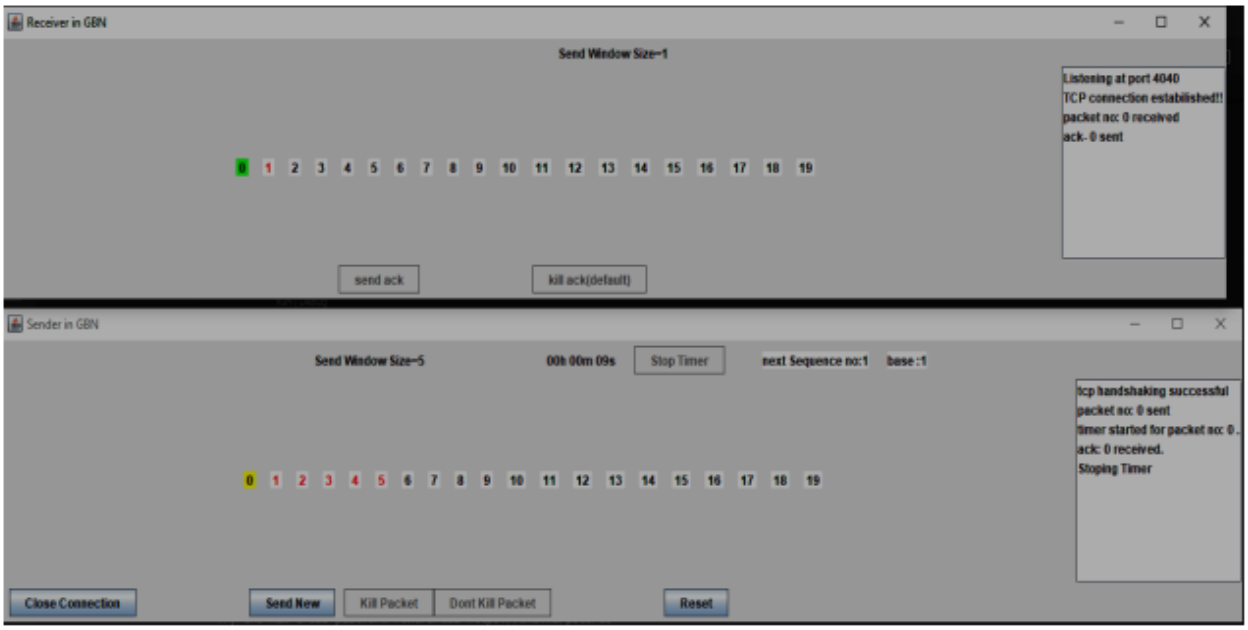
(The Timmer starts, for packet-0, but as it is lost in network, it will not reach recevier)

- When “Don’t Kill Packet” is Pressed in Sender Side:



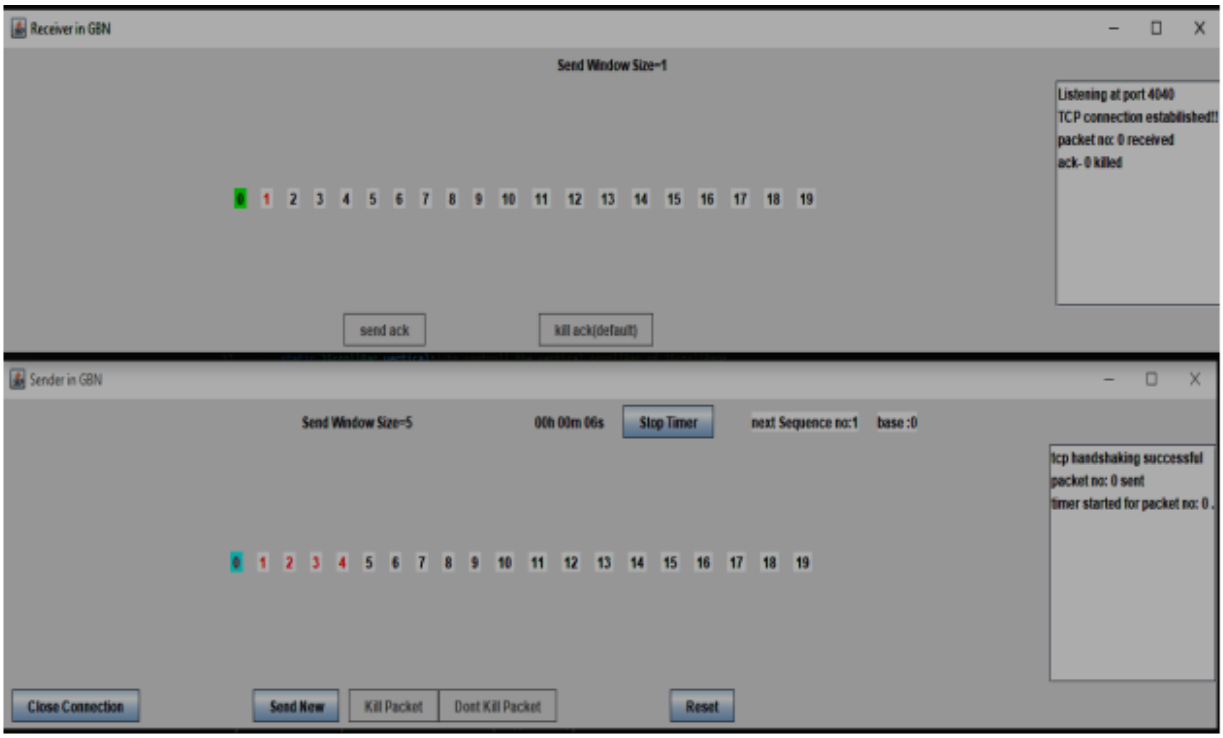
(Timer starts for packet-0, and also packet 0,successfully reaches receiver side)

- When “send ack” is pressed in Receiver Side:



(a acknowledgement(ACK-0) is sent by receiver side, which successfully reaches sender side, and timer is stopped(as no other pending ACK))

• When “Kill ack” is pressed in Receiver Side:



(the ack-0 , sent by receiver side, gets lost in network, and will not reach sender side)

• When Timeout happens, due to ACK getting lost/(overly delayed) in network:

(in the above case, time Timer became 00.00.00 from 00.00.15)

Case 2: Explaining the advantage we get because of PIPELINING:

• Without receiving any ACK, a MAX of N packets can be sent ,where N is the window size.

Case 3: Explaining all cases leading to : Go-Back-N getting implemented

• When a packet(s)/ ACK are lost in network/overly delayed, we need to implement the Go-back-N procedure.

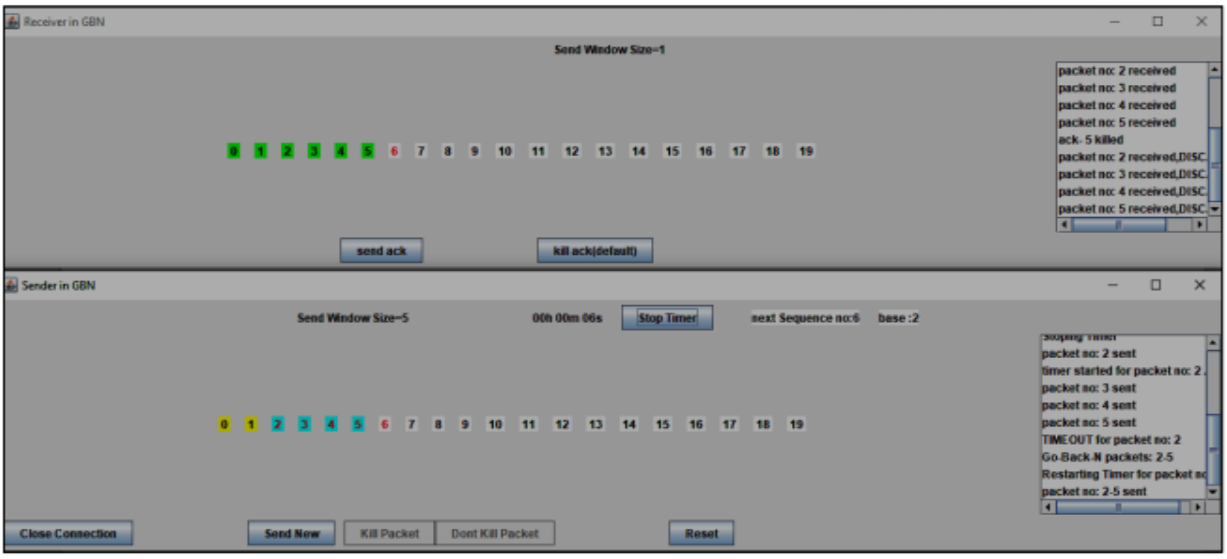
• Consider this situation, when timeout happens for packet [2-5]:

• We implement Go-Back-N for packets [2-5].

• When “Kill Packets[2-5]”, is pressed, it will start the timer, and none of them will reach receiver.

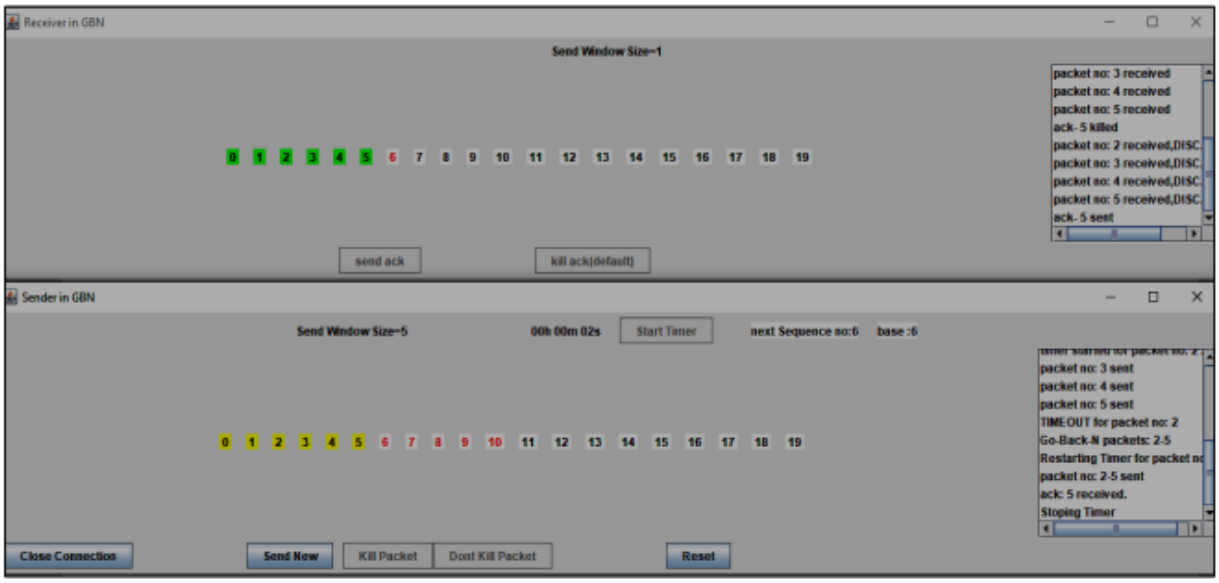
• When “don’t Kill Packets[2-5]” is pressed**, then all packets in [2-5], are sent in sequence, again(duplicate packets), for Receiver.**

• After [2-5] reach receiver:



• Now if we press “send ack”, then a **CUMULATIVE ACKNOWLEDGEMENT,** is sent, for packets [2-5].

• After receiving**, cumulative ack, that is ack(5)** in this case, sent by receiver:



Case 4: After all required data is transferred, the TCP connection is closed:

SR APPLICATION , AT ITS VARIOUS STAGES:

Case 1:

- At the start of the application:

- After TCP connection is created, between sender-receiver:

- When ”Send New Pkt(0) ” is pressed:

(the options now available are: ”Kill packet”, ”Don’t kill packet”)

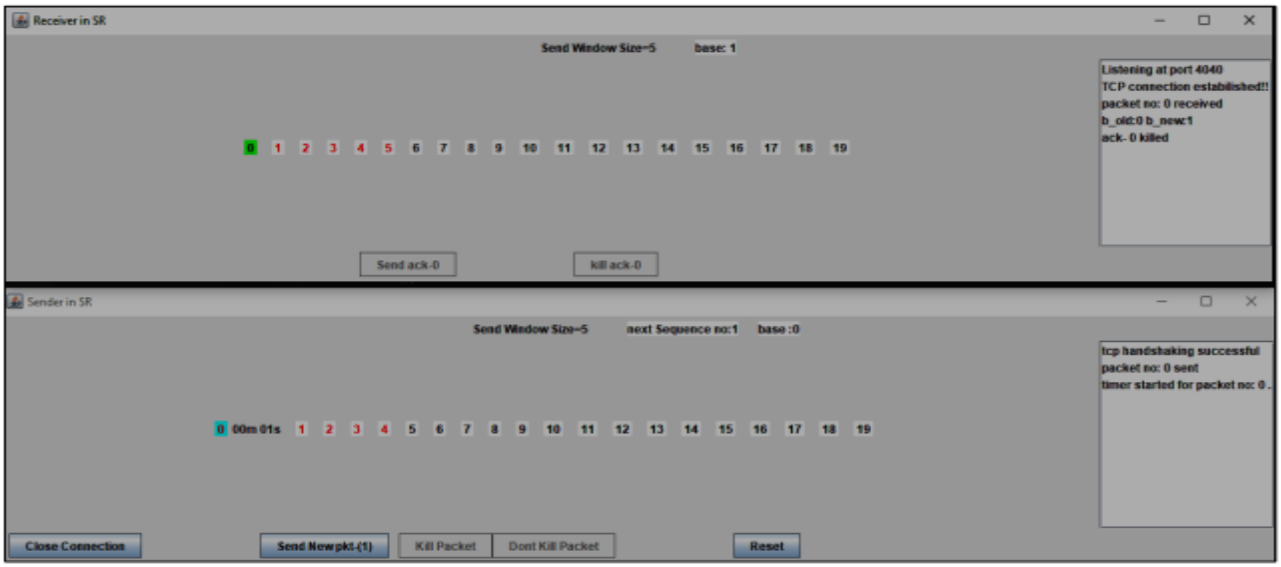
- When “Kill Packet” is pressed, the packet is sent, but is lost in network:

(the timer for packet-0 starts, but it will never reach receiver side, as it is **lost** in network)

• Instead, if “Don’t Kill Packet is pressed”:

(the timer for packet-0 starts, and the packet, successfully reaches receiver side)

• In the above case, when “ kill ack-0” is pressed, in receiver side:



(that is, the sender timer continues, till its timed out, as ack-0 will not reach, as it is **lost/killed** in network)

• Instead if the “send ack-0” is pressed, on the receiver side:

(The timer for packet 0 stops, when ack-0 is received by sender side. The window of Sender Side, slides to the next set of 5 packets, that is [1-5] is the new window for sender side now)

Case 2: Explaining the advantage we get because of PIPELINING:

• A max of N packets(N= window size), can be sent, without getting any ACK, once the limit is reached: “send New pkt” button is disabled.

• In the picture shown below, packets with seq number**: [0-4], are sent, but no ACK is received.**

• The sender, will maintain a timer, for each of these packet, and will resend them, when their timers timeout, till their **ACK is received**.

• The sender side window will slide, after receiving ack-0, that is the ACK **for the base 52 packet, in the window**

Case 3: Explaining all cases leading to : SELECTIVE REPEAT getting implemented

- When a packet is lost/Overly Delayed, timeout happens for that packet

- Also when, the ACK for that particular packet, is lost/ Overly Delayed, timeout will happen

- Consider the following case:

Packet:[0] and packet:[2] are received and acknowledged by receiver side.(ack0,ack2 also successfully reached sender side).But packet 1, is lost in network, and not received by receiver side.

- Timeout happens for packet-1:

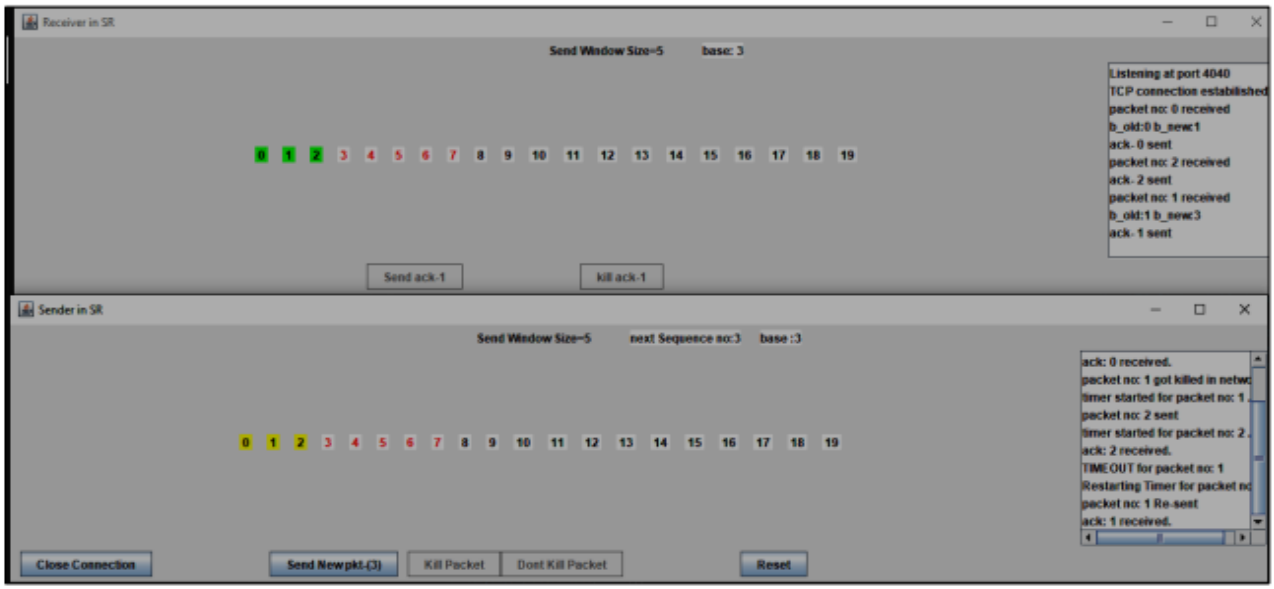
Packet 1, timer , timed out. Now **SELECTIVE REPEAT PROCEDURE will be invoked for packet-1**, and is thus re-sent, by sender side.

- If we select button “Kill Re Sent Packet-1”, then it will wait, again for 15 seconds, before getting Timed out again.

- If we select Button “Don’t Kill Re Sent Packet-1”, then the situation will be:

Packet-1 successfully reaches Receiver side. And we have, 2 options now , On the receiver side

- If “Send ack-1” is selected, ack-1 will successfully reach sender, and the situation will look like:



- If instead “kill ack-1” is selected, then the situation will be

• The sender side, timer for packet-1 will time out, and packet-1 will be resent. Suppose it successfully reaches, the receiver side:

• Receiver will consider, it as a DUPLICATE PACKET, and thus the acknowledgement, for it will be called as “DUPLICATE ACK”.

• If we press button “Kill Duplicate ack-1”, then again the sender side timer for packet-1, will timeout, and above already mentioned steps will be repeated.

• If we press button “Send Duplicate ack-1”, then the situation will be:

Case 4: After all required data is transferred, the TCP connection is closed:

- **“closing socket”** comment, indicates that the channel is closed.

CONCLUSIONS AND FUTURE PLANS

TABLE NO 1:

Summary of Reliable Data Transfer Mechanisms:

MEATHOD	USAGE
Checksum	Used to detect errors, within the data bits, in a packet. Example flipping of 1 to 0,vice versa.
Timer	Used to detect if a packet is **lost/overly delayed.** Timeout will also happen if, ACK for a packet is lost in network.
Sequence number	It was basically introduced, to help a receiver identify if a received packet is a **RE-TRANSMISSION**.(that is to detect /deal with **DUPLICATE** packets.)
Acknowledgement (ACK)	Used by receiver, to give feedback about a single/ group of correctly received packets(without errors). ACK may be individual or cumulative.(depending on protocol).
Negative Acknowledgement (NACK)	NACK is used by receiver to, give feedback about a received **corrupt** packet(sequence number), which needs to be **resent.**
Window of particular Size, Pipelining to utilize available bandwidth	Window size is set upon considering following: 1) Receiver's ability to receive, process, buffer packets(**FLOW CONTROL**) 2) Congestion in network (**CONGESTION CONTROL**)

	Giving sender a opportunity to send more than 1 packet, helps in utilizing the available bandwidth, and exploit the MAX processing capacity of sender.

FUTURE PLANS/IMPROVEMENTS POSSIBLE:

- *Throughout it was assumed that,* ***packet cannot be reordered, in the underlying channel*** *between sender and receiver, however it need not hold true ,always.*

- *When sender, receiver are connected by a single wire, than the above assumption will mostly hold, but if the underlying channel is a network,* ***then packet reordering is possible.***

- *In such a scenario, the sender needs a bit more programming. The sender needs to ensure,before using a particular SEQUENCE NUMBER, that the packet, previously sent with that sequence number, no longer exists in network.*

- *This is achieved by adding a additional field in packet header.(this is done by network layer).*

- *These are:*

  1) In IPv4: it is TIME TO LIVE(TTL), field

  2) In IPv6: it is HOP LIMIT field

- *These fields are decremented , when the packet passes through any intermediate router, when these field becomes 0, the packet is DROPPED.*

• *These ensures that a packet, does not move around INDEFINITELY.*

• *As already mentioned, these headers are included on the segment, in **network layer**, thus it was not shown in GBN,SR protocol application.*

THANK-

- YOU !!!!
