EX.NO: 04	Implementation of Sliding window protocol and stop and wait protocol.
DATE:	

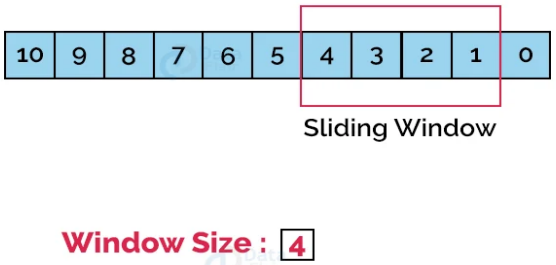
AIM

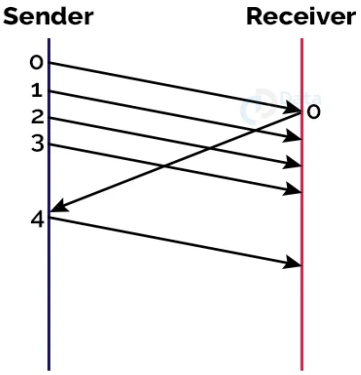
To implement Sliding Window Protocol and Stop and Wait Protocol using Java.

BACKGROUND THEOREY:

➢ Sliding window protocol:

The sliding window protocol is a fundamental method in computer networking for managing data flow between networked devices. It facilitates efficient and reliable transmission over communication channels prone to delays or losses. Flow control mechanisms regulate the size of these windows, preventing overwhelming the receiver or the network. As packets are received, acknowledgments are sent back to the sender, allowing both windows to slide forward. Retransmission ensures reliability by re-sending lost packets after a certain timeout period. Two main variants, Selective Repeat and Go-Back-N, offer different approaches to handling lost or damaged packets. Overall, sliding window protocols are critical for optimizing data transmission and ensuring robust communication in modern computer networks.

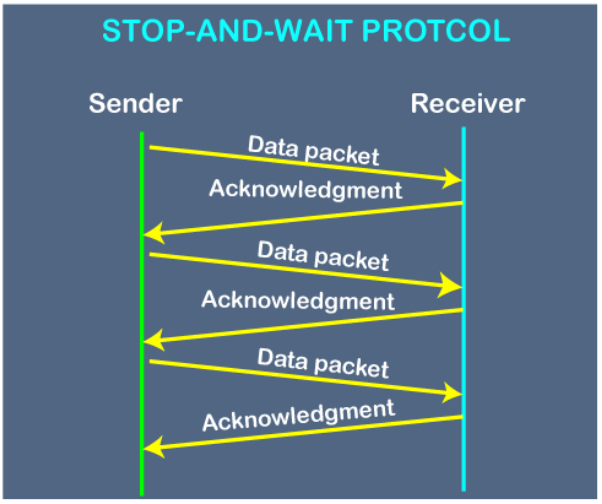




➢ Stop and wait protocol:

The stop-and-wait protocol is a simple method used in computer networking to ensure reliable data transmission between two connected devices. Unlike the sliding window protocol, which allows multiple packets to be transmitted before receiving acknowledgments, stop-and-wait operates by sending one packet at a time and waiting for an acknowledgment before sending the next packet. This method is straightforward and easy to implement but can be less

efficient compared to sliding window protocols, especially over high-latency or unreliable networks. Upon receiving a packet, the receiver sends back an acknowledgment to the sender, indicating successful receipt. If the sender does not receive an acknowledgment within a specified timeout period, it retransmits the same packet. While stop-and-wait ensures reliability, its throughput may be limited by the round-trip time between sender and receiver and the processing time required for each packet. Despite its simplicity, stop-and-wait remains a useful protocol for scenarios where reliability is paramount and network conditions allow for its slower pace of data transmission.



IMPLEMENTATION CODE FOR SLIDING WINDOW:

Sender:

```
import java.net.*;
import java.io.*;
import java.rmi.*;
public class
slidsender
{
public static void main(String a[])throws Exception

{
ServerSocket ser=new
ServerSocket(10); Socket
s=ser.accept();
DataInputStream in=new DataInputStream(System.in);
DataInputStream in1=new
DataInputStream(s.getInputStream()); String sbuff[]=new
String[8];
PrintStream p;
int
sptr=0,sws=8,nf,ano,i;
String ch;
```

```
do
{
p=new
PrintStream(s.getOutputStream());
System.out.print("Enter the no. of frames
: "); nf=Integer.parseInt(in.readLine());
p.println(nf); if(nf<=sws-1)
{
System.out.println("Enter "+nf+" Messages to be
send\n"); for(i=1;i<=nf;i++)
{
sbuff[sptr]=in.readLine
();
p.println(sbuff[sptr]);
sptr=++sptr%8;
}
sws-=nf;
System.out.print("Acknowledgment
received");
ano=Integer.parseInt(in1.readLine());
System.out.println(" for "+ano+" frames");
sws+=nf;
}
else
{
System.out.println("The no. of frames exceeds window size");
break;
}
System.out.print("\nDo you wants to send some more
frames : "); ch=in.readLine(); p.println(ch);
}
while(ch.equals("yes"
)); s.close();
}
}
```

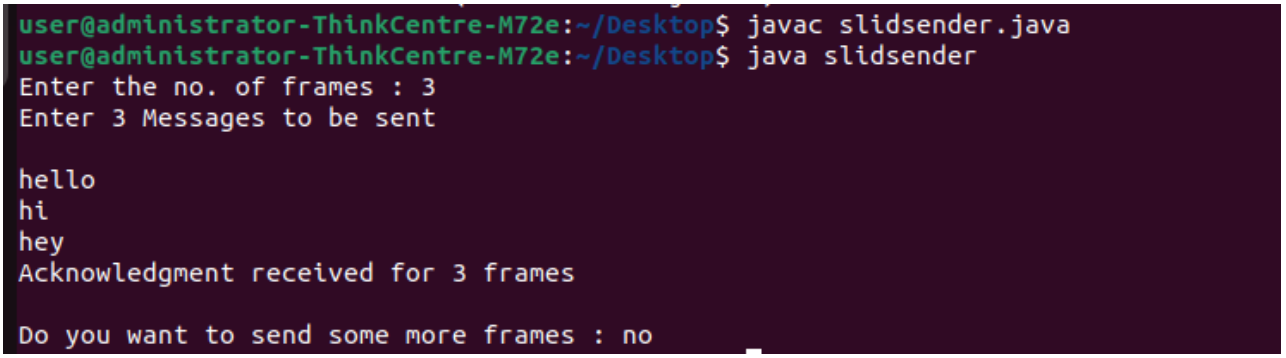
Receiver:

```
import java.net.*; import java.io.*; class slidreceiver
{
public static void main(String a[])throws Exception
{
Socket s=new Socket(InetAddress.getLocalHost(),10);
DataInputStream in=new
DataInputStream(s.getInputStream()); PrintStream p=new
PrintStream(s.getOutputStream());
int i=0,rptr=-1,nf,rws=8;
String rbuf[]=new String[8];
String ch;
System.out.println(); do
{
nf=Integer.parseInt(in.readLine
()); if(nf<=rws-1)
```

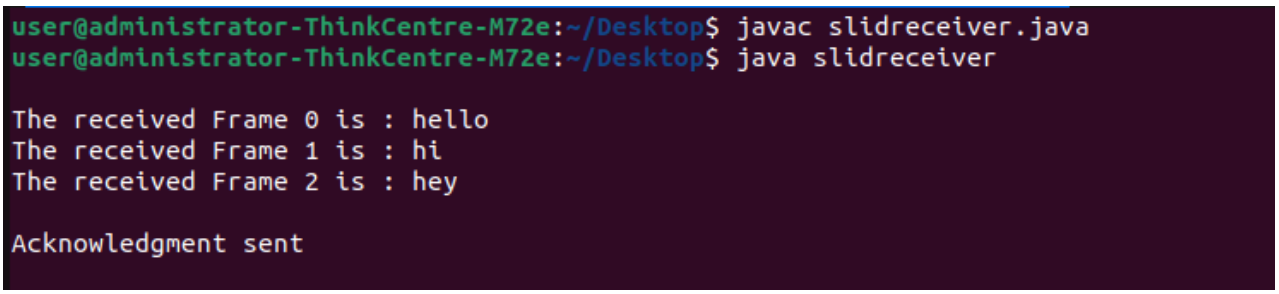
```
{
for(i=1;i<=nf;i++){
rptr=++rptr%8;
rbuf[rptr]=in.readLine
();
System.out.println("The received Frame " +rptr+" is : "+rbuf[rptr]);
}
rws-=nf;
System.out.println("\nAcknowledgment
sent\n"); p.println(rptr+1); rws+=nf; }
else
break
;
ch=in.readLine();
}
while(ch.equals("yes"));
}
}
```

OUTPUT

Sender:



Receiver:



IMPLEMENTATION CODE FOR STOP AND WAIT:

Sender:

```
import java.io.*;
import java.net.*;
public class
Sender{ Socket
sender;
ObjectOutputStream
out; ObjectInputStream
in; String packet,ack,str,
msg; int
n,i=0,sequence=0;
Sender(){}
public void
 run(){ try{
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in)); System.out.println("Waiting for
Connection.                                    ");
 sender = new
Socket("localhost",2004);
sequence=0;
out=new
ObjectOutputStream(sender.getOutputStream());
out.flush();
 in=new
ObjectInputStream(sender.getInputStream());
str=(String)in.readObject();
System.out.println("receiver > "+str);
System.out.println("Enter the data to send.  ");
packet=br.readLine
();
n=packet.length();
do{
try{
 if(i<n)
 {
msg=String.valueOf(sequence);
msg=msg.concat(packet.substring(i,i+1));}
else if(i==n){
 msg="end";out.writeObject(msg);break;
}
out.writeObject(msg);
sequence=(sequence==0)?1
:0; out.flush();
System.out.println("data sent>"+msg);
back=(String)in.readObject(); System.out.println("waiting for ack.          \n\n");
if(ack.equals(String.valueOf(sequence)))
{ i++;
System.out.println("receiver > "+" packet recieved\n\n");
}
```

```
else{
System.out.println("Time out resending data.          \n\n");
sequence=(sequence==0)?1:0;
}
}catch(Exception e){}
}while(i<n+1);
System.out.println("All data sent. exiting.");
}catch(Exception
e){} finally{
try{
in.close(
);
out.close();
sender.close(
);
}
catch(Exception e){}
}
}
public static void main(String
args[]){ Sender s=new Sender();
s.run();
}
}
```

Receiver:

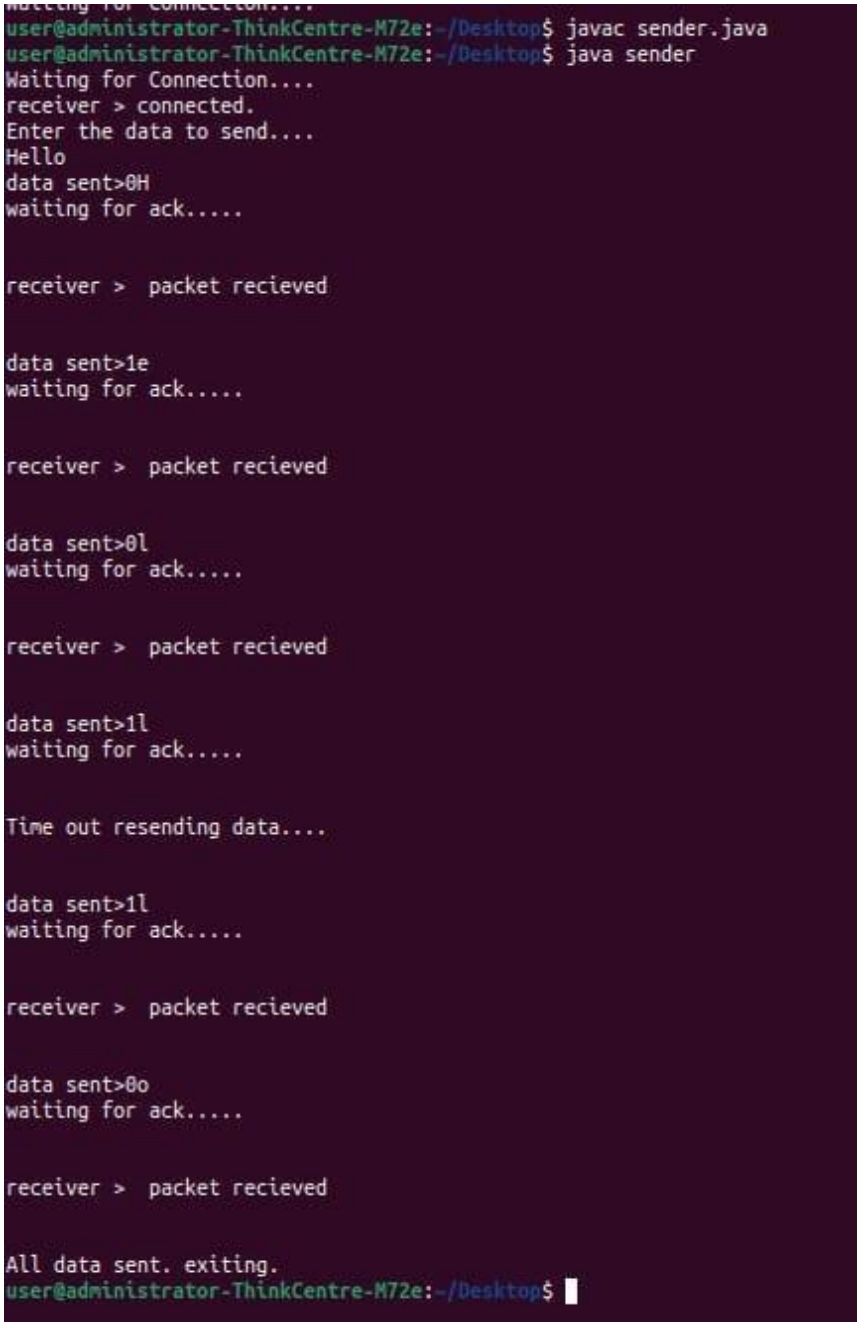
```
import java.io.*; import java.net.*; public class Receiver{ ServerSocket reciever;
Socket
connection=null;
ObjectOutputStream
out; ObjectInputStream
in; String
packet,ack,data=""; int
i=0,sequence=0;
Reciever(){}
public void
run(){ try{
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in)); reciever = new ServerSocket(2004,10);
System.out.println("waiting for connection. ");
connection=reciever.accept
(); sequence=0;
System.out.println("Connection established :");
out=new
ObjectOutputStream(connection.getOutputStream());
out.flush();
in=new

ObjectInputStream(connection.getInputStream());
out.writeObject("connected  .");
do{
try
{
```

```
packet=(String)in.readObject();
if(Integer.valueOf(packet.substring(0,1))==sequence){
data+=packet.substring(1); sequence=(sequence==0)?1:0;
System.out.println("\n\nreceiver >"+packet);
}
else
{
System.out.println("\n\nreceiver>"+packet +" duplicate data");
}
if(i<3){
out.writeObject(String.valueOf(sequence));i++;
}
else{
out.writeObject(String.valueOf((sequence+1)%
2)); i=0;
}
}
catch(Exception e){}
}while(!packet.equals("end"));
System.out.println("Data
recived="+data);
out.writeObject("connection ended
.");
}
catch(Exception
e){} finally{
try{
in.close(
);
out.close();
reciever.close(
);
}
catch(Exception e){}}
}
public static void main(String
args[]){ Reciever s=new
Reciever(); while(true){
s.run();
}}}
```

SAMPLE OUTPUT

Sender:



Receiver:

```
user@administrator-ThinkCentre-M72e:~/Desktop$ javac Receiver.java
user@administrator-ThinkCentre-M72e:~/Desktop$ java Receiver
waiting for connection...
Connection established:


receiver >0H


receiver >1e


receiver >0l


receiver >1l


receiver>1l duplicate data


receiver >0o
Data received=Hello
```

RESULT

Thus the java program for implementing stop & wait and sliding window protocol is created and executed.