University of Dhaka

Assignment on Implementing Reliable File Transfer over an Unreliable Transport

Md.Al-Helal(Roll-51)
Md.Rashid Ahmed(Roll-15)

submitted to
Dr. Md. Mamun-or-Rashid
Professor
Dr. Shabbir Ahmed
Professor and Chairperson

December 5, 2017

Theory

There are several types of protocol for reliable data transfer. We have used *Selective Repeat ARQ* here to implement reliable data transfer using datagram socket (UDP socket). In this protocol, if sender has window size of n, receiver has to be window size n/2. Window size is the number sender can send to receiver without any acknowledgement. After sending n (window size) packets to receiver, it waits for receiving acknowledgement from receiver. If any packet from sender is lost or corrupted, same packet is resent to receiver for reliability. If lost, packet is retransmitted after a certain time which is greater than 4 times *rtt* (round trip time). We can find this *rtt* by *ping* command. Receiver sends acknowledgement based on received packet. If the packet is corrupted, it sends negative acknowledgement and waits for receiving it again. Then sender send the same packet again corresponding the negative acknowledgement.

Source Code

Server Application

```
import java.net.*;

public class SenderApplication
{
    public static void main(String[] args)
    {
        try
        {
            String hiMessage = "Hi";
            DatagramSocket serverSocket = new DatagramSocket(3333);
            System.out.println("Waiting to be connected.");
            InetAddress clientIPaddress = InetAddress.getByName("localhost"
            int clientPort = 2222;

            DatagramPacket sendPacket = new DatagramPacket(hiMessage.getBy
            serverSocket.send(sendPacket);
            System.out.println("Connected.");

            AcknowledgementReceiver ackReceiver = new AcknowledgementRecei
            ackReceiver.receiveAck();

        }
        catch (Exception ex)
        {
            System.out.println(ex);
        }
    }
```

```
}

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.Timer;
import java.util.TimerTask;

public class SenderThread implements Runnable
{
    private DatagramPacket packet;
    private final DatagramSocket serverSocket;
    private final Timer timer;
    public Thread thread;
    private final PacketGenerator packetGenerator;

    public SenderThread(DatagramSocket serverSocket)
    {
        this.serverSocket = serverSocket;
        timer = new Timer();
        thread = new Thread(this);
        packetGenerator = PacketGenerator.getInstance();
    }

    public void setNewPacket()
    {
        packet = packetGenerator.getPacket();
    }

    @Override
    public void run()
    {
        while (true)
        {
            try
            {
                if (packet == null)
                {
                    String byeMessage = "Good bye";
                    packet = new DatagramPacket(byeMessage.getBytes(), byeM
                    serverSocket.send(packet);
                    break;
                }
                serverSocket.send(packet);
            }
            catch (Exception e)
```

```
            {
                System.out.println(e);
            }
            timer.scheduleAtFixedRate(new TimerTask()
            {
                @Override
                public void run()
                {
                    try
                    {
                        serverSocket.send(packet);
                    }
                    catch (IOException ex)
                    {
                        System.out.println(ex);
                    }
                }
            }, 200, 200);
        }
    }
}

import java.io.*;
import java.net.DatagramPacket;

public class PacketGenerator
{
    private volatile static PacketGenerator uniqueInstance;
    private BufferedReader inputBuffer;
    private int packetDataSize, packetSize, currentSequenceNumber, windowS

    private PacketGenerator()
    {
        inputBuffer = null;
        packetDataSize = 10;
        packetSize = 12;
        currentSequenceNumber = 0;
        windowSize = 10;
        try
        {
            inputBuffer = new BufferedReader(new FileReader(new File("text.
        }
        catch (FileNotFoundException ex)
        {
            System.out.println(ex);
        }
```

```
}

public static PacketGenerator getInstance()
{
    if (uniqueInstance == null)
    {
        synchronized (PacketGenerator.class)
        {
            if (uniqueInstance == null)
            {
                uniqueInstance = new PacketGenerator();
            }
        }
    }
    return uniqueInstance;
}

private DatagramPacket generatePacket()
{
    try
    {
        char[] packetData = new char[packetSize];
        inputBuffer.read(packetData, 0, packetDataSize);
        addParity(packetData);
        packetData[11] = (char) currentSequenceNumber;
        currentSequenceNumber = (currentSequenceNumber + 1) % windowSiz
        return new DatagramPacket(new String(packetData).getBytes(), pa
    }
    catch (IOException ex)
    {
        System.out.println(ex);
    }
    return null;
}

public DatagramPacket getPacket()
{
    return generatePacket();
}

private void addParity(char[] packetData)
{
    int count = 0;
    for (int i = 0; i < 10; i++)
    {
        if (packetData[i] == '1')
```

```
            {
                count++;
            }
        }
        packetData[10] = count % 2 == 0 ? '0' : '1';//Event parity.
    }
}

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class AcknowledgementReceiver
{
    private final SenderThread[] senderThreads;
    private final DatagramSocket serverSocket;
    private DatagramPacket clientPacket;
    int windowSize = 10;
    int startIndex, endIndex;

    public AcknowledgementReceiver(DatagramSocket serverSocket)
    {
        this.serverSocket = serverSocket;
        senderThreads = new SenderThread[windowSize];
        startIndex = 0;
        endIndex = windowSize - 1;
        for (int i = 0; i < 10; i++)
        {
            senderThreads[i] = new SenderThread(serverSocket);
            senderThreads[i].thread.start();
            //receiveAck();
        }
    }

    public void receiveAck()
    {
        try
        {
            while (true)
            {
                serverSocket.receive(clientPacket);
                if (clientPacket == null)
                {
                    break;
                }
                String sequenceNo = new String(clientPacket.getData(), 0, c
```

```
                int ackNumber = Integer.parseInt(sequenceNo);
                while (startIndex != ackNumber)
                {
                    senderThreads[startIndex].setNewPacket();
                    startIndex = (startIndex + 1) % windowSize;
                    endIndex = (endIndex + 1) % windowSize;
                }
            }
        }
        catch (IOException ex)
        {
            System.out.println(ex);
        }
    }
}
```

Receiver Application

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

public class ReceiverApplication
{
    static InetAddress clientIPaddress;
    static int clientPort = 2222;

    public static void main(String[] args)
    {
        try
        {
            byte[] receiveBytes = new byte[8];
            DatagramSocket serverSocket = new DatagramSocket(clientPort);
            DatagramPacket receivePacket = new DatagramPacket(receiveBytes,
            handShaking(serverSocket, receivePacket);
            PacketReceiver packetReceiver = PacketReceiver.getInstance(serv
            packetReceiver.storePacket(receivePacket);
        }
        catch (SocketException ex)
        {
            System.out.println(ex);
        }
    }

    private static void handShaking(DatagramSocket socket, DatagramPacket p
    {
```

```
        try
        {
            System.out.println("Waiting for receiving a packet.");
            socket.receive(packet);
            clientIPaddress = packet.getAddress();
            clientPort = packet.getPort();
            System.out.println("Received: " + new String(packet.getData(),
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
    }
}

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.Timer;

public class ReceiverThread implements Runnable
{
    private DatagramPacket packet;
    private final DatagramSocket serverSocket;
    private final Timer timer;
    public Thread thread;
    private final PacketReceiver packetStore;

    public ReceiverThread(DatagramSocket serverSocket)
    {
        this.serverSocket = serverSocket;
        timer = new Timer();
        thread = new Thread(this);
        packetStore = PacketReceiver.getInstance(serverSocket);
    }

    @Override
    public void run()
    {
        while (true)
        {
            try
            {
                serverSocket.receive(packet);
                String byeMessage = new String(packet.getData(), 0, packet.
```

```
                if (byeMessage.equals("Good bye"))
                {
                    break;
                }
                packetStore.storePacket(packet);
            }
            catch (IOException ex)
            {
                System.out.println(ex);
            }
        }
    }
}

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class PacketReceiver
{
    private final ReceiverThread[] receiverThread;
    private final DatagramSocket serverSocket;
    private volatile static PacketReceiver uniqueInstance;
    private DatagramPacket clientPacket;
    int windowSize = 10;
    int startIndex, endIndex;
    private DatagramPacket[] packetList;

    private PacketReceiver(DatagramSocket serverSocket)
    {
        this.serverSocket = serverSocket;
        receiverThread = new ReceiverThread[windowSize];
        startIndex = 0;
        endIndex = windowSize - 1;
        for (int i = 0; i < 10; i++)
        {
            receiverThread[i] = new ReceiverThread(serverSocket);
            receiverThread[i].thread.start();
        }
    }

    public static PacketReceiver getInstance(DatagramSocket serverSocket)
    {
        if (uniqueInstance == null)
        {
            synchronized (PacketReceiver.class)
```

```
        {
            if (uniqueInstance == null)
            {
                uniqueInstance = new PacketReceiver(serverSocket);
            }
        }
    }
    return uniqueInstance;
}

private boolean isErrorPacket(DatagramPacket packet)
{
    String data = new String(packet.getData(), 0, packet.getLength()).s
    int counter = 0;
    for (int i = 0; i < data.length(); i++)
    {
        if (data.charAt(i) == '1')
        {
            counter++;
        }
    }
    return counter % 2 != 0;
}

public void storePacket(DatagramPacket packet)
{
    String data = new String(packet.getData(), 0, packet.getLength());
    int packetIndex = data.charAt(11) - '0';
    packetList[packetIndex] = packet;

    while (startIndex != packetIndex)
    {
        if (!isErrorPacket(packet))
        {
            startIndex = (startIndex + 1) % windowSize;
            endIndex = (endIndex + 1) % windowSize;
        }
        else
        {
            try
            {
                String ack = String.format("%d", startIndex);
                DatagramPacket ackPacket = new DatagramPacket(ack.getB
                serverSocket.send(ackPacket);
            }
            catch (IOException ex)
```

```
                {
                    System.out.println(ex);
                }
            }
        }
    }
}
```

Sample Output

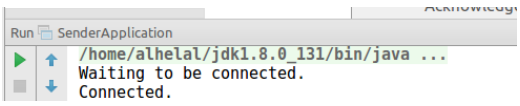

Figure 1: Sender application

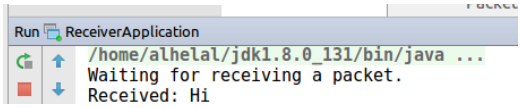

Figure 2: Receiver application

Problem encountered during implementation

- We have used thread to send multiple packets to receiver. Monitoring and handing multhreading was difficult to us.
- After sending each packet, *timer* has to be started. And receiving an acknowledgement reinitializing the corresponding thread with a new packet is also difficult. Designing timer was problematic to us.
- Writing programs need to monitoring each single points. For this, our observation power has to be enhanced.

Conclusion

Designing reliable data transfer is some combination of small and simple task. But good combination can enhance its ability. Using datagram socket (UDP socket) is not reliable, but we have designed a reliable data transfer protocol using this unreliable service.