```java
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

class Packet {
    int sequenceNumber;
    String data;

    Packet(int sequenceNumber, String data) {
        this.sequenceNumber = sequenceNumber;
        this.data = data;
    }
}

public class FrameSort {
    private static final int FRAME_SIZE = 3;

    public static Packet[] divideMessageIntoPackets(String message) {
        int messageLength = message.length();
        int numberOfPackets = (int) Math.ceil((double) messageLength / FRAME_SIZE);
        Packet[] packets = new Packet[numberOfPackets];

        for (int i = 0, j = 0; i < numberOfPackets; i++) {
            int end = Math.min(j + FRAME_SIZE, messageLength);
            packets[i] = new Packet(i + 1, message.substring(j, end));
            j += FRAME_SIZE;
        }

        return packets;
    }

    public static Packet[] shufflePackets(Packet[] originalPackets) {
        Random random = new Random();
        Packet[] shuffledPackets = Arrays.copyOf(originalPackets,
originalPackets.length);

        for (int i = 0; i < shuffledPackets.length; i++) {
            int randomIndex = random.nextInt(shuffledPackets.length);
            Packet temp = shuffledPackets[i];
            shuffledPackets[i] = shuffledPackets[randomIndex];
            shuffledPackets[randomIndex] = temp;
        }

        return shuffledPackets;
    }

    public static void bubbleSortPackets(Packet[] packets) {
        int n = packets.length;
        boolean swapped;
        do {
            swapped = false;
            for (int i = 1; i < n; i++) {
                if (packets[i - 1].sequenceNumber > packets[i].sequenceNumber) {
                    Packet temp = packets[i - 1];
                    packets[i - 1] = packets[i];
                    packets[i] = temp;
                    swapped = true;
                }
            }
            n--;
        } while (swapped);
    }

    public static void receiveAndProcessPackets(Packet[] receivedPackets) {
        System.out.println("\nPackets received in the following order:");
        for (Packet packet : receivedPackets) {
            System.out.printf("Frame %d: %s%n", packet.sequenceNumber, 
packet.data);
        }

        bubbleSortPackets(receivedPackets);

        System.out.println("\n\nPackets in order after sorting:");
        for (Packet packet : receivedPackets) {
            System.out.printf("Frame %d: %s%n", packet.sequenceNumber, 
packet.data);
        }

        System.out.println("\n\nMessage received is :\n");
        StringBuilder reconstructedMessage = new StringBuilder();
        for (Packet packet : receivedPackets) {
            reconstructedMessage.append(packet.data);
        }
        System.out.println(reconstructedMessage);
    }

    public static void main(String[] args) {
        String message;
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the message to be transmitted:");
        message = scanner.nextLine();

        System.out.println("\nMessage divided into frames:");
        Packet[] originalPackets = divideMessageIntoPackets(message);
        for (Packet packet : originalPackets) {
            System.out.printf("Frame %d: %s%n", packet.sequenceNumber, 
packet.data);
        }
        Packet[] shuffledPackets = shufflePackets(originalPackets);
        receiveAndProcessPackets(shuffledPackets);

        scanner.close();
    }
}


/*Enter The message to be Transmitted:
My name is Ashwath

Message divided into frames:
Frame 1: My
Frame 2: nam
Frame 3: e i
Frame 4: s A
Frame 5: shw
Frame 6: ath

Packets received in the following order:
Frame 3: e i
Frame 2: nam
Frame 4: s A
Frame 1: My
Frame 5: shw
Frame 6: ath


Packets in order after sorting:
Frame 1: My
Frame 2: nam
Frame 3: e i
Frame 4: s A
Frame 5: shw
Frame 6: ath


Message received is :

My name is Ashwath */
```

```
import java.util.Scanner;

public class lab3_1 {
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the no of bits : ");
        int n = s.nextInt();

        int data[] = new int[n];

        System.out.println("Enter the data bits : ");
        for (int i = 0; i < n; i++)
            data[i] = s.nextInt();

        System.out.println("Enter the no of divisor bits : ");
        int m = s.nextInt();
        int divisor[] = new int[m];

        System.out.println("Enter divisor bits : ");
        for (int j = 0; j < m; j++)
            divisor[j] = s.nextInt();

        int len = n + m - 1;
        int div[] = new int[len];
        int rem[] = new int[len];
        int crc[] = new int[len];

        for (int i = 0; i < data.length; i++)
            div[i] = data[i];

        System.out.println("Dividend after appending zero");
        for (int i = 0; i < div.length; i++)
            System.out.print(div[i]);
        System.out.println();

        for (int j = 0; j < div.length; j++)
            rem[j] = div[j];

        rem = divide(div, divisor, rem);

        for (int i = 0; i < div.length; i++)
            crc[i] = (div[i] ^ rem[i]);

        System.out.println();
        System.out.println("CRC code");

        for (int i = 0; i < crc.length; i++)
            System.out.print(crc[i]);
        System.out.println();
        System.out.println("CRC code of " + len + " bits");
        System.out.println("Enter CRC code");
        for (int i = 0; i < crc.length; i++)
            crc[i] = s.nextInt();

        for (int i = 0; i < len; i++)
            rem[i] = crc[i];

        rem = divide(crc, divisor, rem);

        boolean isError = false;
        for (int i = 0; i < rem.length; i++) {
            if (rem[i] != 0) {
                isError = true;
                break;
            }
        }

        if (isError) {
            System.out.println("Error detected!");
        } else {
            System.out.println("No error detected.");
        }

        s.close();
    }

    static int[] divide(int div[], int divisor[], int rem[]) {
        int cur = 0;
        while (true) {
            for (int i = 0; i < divisor.length; i++)
                rem[cur + i] = (rem[cur + i] ^ divisor[i]);

            while (rem[cur] == 0 && cur != rem.length - 1)
                cur++;

            if ((rem.length - cur) < divisor.length)
                break;
        }
        return rem;
    }
}
/*Enter the no of bits :
16
Enter the data bits :
1 0 1 1 0 0 1 0 0 0 1 1 0 0 1 0
Enter the no of divisor bits :
4
Enter divisor bits :
1 0 0 1
```

```
Dividend after appending zero
1011001000110010

CRC code
1100

CRC code of 19 bits
Enter CRC code
1011001000110010

No error detected.

*/
```

```
import java.util.Arrays;
import java.util.Scanner;

public class BellmanFord{
        public static int n;
        private static int[][] graph;
        public static void bellmanford(int src){
                int[] dist=new int[n];
                Arrays.fill(dist,Integer.MAX_VALUE);
                dist[src]=0;
                for(int i=1;i<n;i++){
                        for(int u=0;u<n;u++){
                                for(int v=0;v<n;v++){

if(graph[u][v]!=0&&dist[u]!=Integer.MAX_VALUE&&dist[u]+graph[u][v]<dist[v]){
                                                dist[v]=dist[u]+graph[u][v];
                                        }
                                }
                        }
                }
                for(int u=0;u<n;u++){
                        for(int v=0;v<n;v++){

if(graph[u][v]!=0&&dist[u]!=Integer.MAX_VALUE&&dist[u]+graph[u][v]<dist[v]){
                                        System.out.println("Negative weight cycle
detected");
                                        return;
                                }
                        }
                }
                printSolution(dist);
        }
        public static void printSolution(int[] dist){
                System.out.println("Vertex\t Distance from source");
                for(int i=0;i<n;i++)
                        System.out.println((i+1)+"\t\t"+dist[i]);
        }
        public static void main(String[] args){
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter number of vertices:");
                n=sc.nextInt();
                System.out.println("Enter the weight Matrix of Graph");
                graph=new int[n][n];
                for(int i=0;i<n;i++)
                        for(int j=0;j<n;j++)
                                graph[i][j]=sc.nextInt();
                System.out.print("Enter source vertex:");
                int source=sc.nextInt();
                bellmanford(source-1);
        }
}
/*Enter number of vertices:5
Enter the weight Matrix of Graph
0 -1 4 1000 1000
1000 0 3 2 2
1000 1000 0 1000 1000
1000 1 5 0 1000
1000 1000 1000 -3 0
Enter source vertex:1
Vertex   Distance from source
1               0
2               -1
3               2
4               -2
5               1 */
```

```
//TcpClient.java
import java.net.*;
import java.io.*;

public class TcpClient {
    public static void main(String[] args) throws Exception {
        Socket sock = new Socket("127.0.0.1", 4000); //change port no. to 4001 if
already in use
        System.out.println("Enter the file name: ");
        BufferedReader nameRead = new BufferedReader(new
InputStreamReader(System.in));
        String fname = nameRead.readLine();
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        pwrite.println(fname);

        InputStream istream = sock.getInputStream();
        BufferedReader contentRead = new BufferedReader(new
InputStreamReader(istream));
        String str;
        while ((str = contentRead.readLine()) != null) {
            System.out.println(str);
        }
        contentRead.close();
        pwrite.close();
        sock.close();
        nameRead.close();
    }
}

//TcpServer.java
import java.net.*;
import java.io.*;

public class TcpServer {
    public static void main(String[] args) throws Exception {
        ServerSocket sersock = new ServerSocket(4000);
        System.out.println("Server Connected, waiting for client");
        Socket sock = sersock.accept();
        System.out.println("Connection successful, waiting for filename");
        InputStream iStream = sock.getInputStream();
        BufferedReader nameRead = new BufferedReader(new
InputStreamReader(iStream));
        String fname = nameRead.readLine();
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);
        try {
            BufferedReader contentRead = new BufferedReader(new FileReader(fname));
            String str;
            while ((str = contentRead.readLine()) != null) {
                pwrite.println(str);
            }
            contentRead.close();
        } catch (FileNotFoundException e) {
            pwrite.println("File does not exist");
        } finally {
            System.out.println("Closing connection");
            pwrite.close();
            nameRead.close();
            sock.close();
            sersock.close();
        }
    }
 }
```

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

class UdpServer
{
        public static void main(String args[]) throws Exception
        {
                Scanner in = new Scanner(System.in);
                DatagramSocket datagramSocket = new DatagramSocket();
                InetAddress clientAddress = InetAddress.getByName("127.0.0.1");
                String message;
                byte[] buffer;
                DatagramPacket datagramPacket;
                System.out.println("Enter messages to send: ");
                while (true){
                        message = in.nextLine();
                        buffer = message.getBytes();
                        datagramPacket = new DatagramPacket(buffer, buffer.length,
clientAddress, 4000);
                        datagramSocket.send(datagramPacket);

                        if (message.equalsIgnoreCase("exit")) {
                                datagramSocket.close();
                                break;
                        }

                }
        }

}

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

class UdpClient
{
        public static void main(String args[]) throws Exception
        {
                DatagramSocket datagramSocket = new DatagramSocket(4000);
                byte[] buffer;
                DatagramPacket datagramPacket;
                System.out.println("Received Messages: ");
                while(true)
                {
                        buffer = new byte[65555];
                        datagramPacket = new DatagramPacket(buffer, buffer.length);
                        datagramSocket.receive(datagramPacket);
                        String received = new String(buffer).trim();
                        System.out.println(received);
```

```
            if (received.equalsIgnoreCase("exit")) {
                datagramSocket.close();
                break;
            }
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class RSACrypto {
    public static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public static int findCoprime(int phi) {
        int e = 2;
        while (e < phi) {
            if (gcd(e, phi) == 1) {
                return e;
            }
            e++;
        }
        return -1; // Handle the case where no coprime is found
    }

    public static int findPrivateKey(int e, int phi) {
        int d = 0;
        int k = 1;
        while (true) {
            double tempD = (1 + k * phi) / (double) e;
            if (tempD == (int) tempD) {
                d = (int) tempD;
                return d;
            }
            k++;
        }
    }

    public static int modularExp(int b, int e, int m) {
        int res = 1;
        for (int i = 0; i < e; i++) {
            res = (res * b) % m;
        }
        return res;
    }

    public static List<Integer> rsaEncrypt(String plaintext, int e, int n) {
        List<Integer> ciphertext = new ArrayList<>();
        for (char c : plaintext.toCharArray()) {
            ciphertext.add(modularExp(c, e, n));
        }
        return ciphertext;
    }

    public static String rsaDecrypt(List<Integer> ciphertext, int d, int n) {
        StringBuilder plaintext = new StringBuilder();
        for (int value : ciphertext) {
            plaintext.append((char) modularExp(value, d, n));
        }
        return plaintext.toString();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int p = 17;
        int q = 23;
        int n = p * q;
        int phi = (p - 1) * (q - 1);

        int e = findCoprime(phi);
        if (e == -1) {
            System.out.println("No coprime found for the given phi.");
            return;
        }

        int d = findPrivateKey(e, phi);

        System.out.print("Enter plaintext: ");
        String plaintext = scanner.nextLine();

        List<Integer> ct = rsaEncrypt(plaintext, e, n);
        System.out.print("Ciphertext: ");
        for (int value : ct) {
            System.out.print(value + " ");
        }
        System.out.println();

        String pt = rsaDecrypt(ct, d, n);
        System.out.println("Plaintext: " + pt);
    }
}
```

```
import java.util.Scanner;

public class LeakyBucket {
    public static void main(String[] args) {
        int noOfQueries = 4;
        int bucketSize = 10;
        int inputPacketSize;
        int outputPacketSize = 1;
        int storedBufferSize = 0;
        int sizeLeft;

        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < noOfQueries; i++) {
            System.out.print("Input Packet Size: ");
            inputPacketSize = scanner.nextInt();

            sizeLeft = bucketSize - storedBufferSize;
            if (inputPacketSize <= sizeLeft) {
                storedBufferSize += inputPacketSize;
            } else {
                System.out.println("Packet Dropped");
            }
            System.out.println("Stored Buffer Size: " + storedBufferSize);
            storedBufferSize -= outputPacketSize;
        }
    }
}
```

```
import java.util.Scanner;

public class TokenBucket {
    public static void main(String[] args) throws InterruptedException {
        int tokens = 0; // initial number of tokens in the bucket
        int rate = 10; // rate at which tokens are added to the bucket
        int capacity = 100; // maximum number of tokens the bucket can hold

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of requests: ");
        int n = scanner.nextInt();

        int[] request = new int[n];
        System.out.print("Enter no. of packets per request: ");
        for (int i = 0; i < n; i++) {
            request[i] = scanner.nextInt();
        }

        for (int i = 0; i < n; i++) {
            // Add tokens to the bucket at a fixed rate
            tokens = Math.min(tokens + rate, capacity);

            // Wait for 1 second
            Thread.sleep(1000);

            if (tokens >= request[i]) {
                // Remove the requested tokens from the bucket
                tokens -= request[i];
                System.out.println("Request granted, tokens remaining: " + tokens);
            } else {
                System.out.println("Request denied, not enough tokens: " + tokens);
            }
        }
    }
}
```

```
import java.util.Random;

public class REDCongestionControl {
    public static void main(String[] args) {
        int MAX_PACKETS = 20;
        int QUEUE_SIZE = 10;
        double MAX_PROBABILITY = 0.7;
        double MIN_PROBABILITY = 0.3;

        Random random = new Random();
        int queueLength = 0;
        double dropProbability = MIN_PROBABILITY;

        for (int i = 0; i < MAX_PACKETS; i++) {
            if (queueLength == QUEUE_SIZE) {
                System.out.println("Packet dropped (QUEUE FULL)");
                dropProbability = MIN_PROBABILITY;
            } else if (random.nextDouble() < dropProbability) {
                System.out.println("Packet dropped (RANDOM)");
                dropProbability += (MAX_PROBABILITY - MIN_PROBABILITY) /
(MAX_PACKETS - 1);
            } else {
                System.out.println("Packet accepted");
                queueLength++;
                dropProbability = MIN_PROBABILITY;
            }
        }
    }
}
```

```
Packet accepted
Packet accepted
Packet dropped (RANDOM)
Packet dropped (RANDOM)
Packet dropped (RANDOM)
Packet accepted
Packet dropped (RANDOM)
Packet accepted
Packet accepted
Packet dropped (RANDOM)
Packet accepted
Packet accepted
Packet accepted
Packet accepted
Packet accepted
Packet dropped (QUEUE FULL)
Packet dropped (QUEUE FULL)
Packet dropped (QUEUE FULL)
Packet dropped (QUEUE FULL)
Packet dropped (QUEUE FULL)
```

```
import java.util.Random;
import java.util.Scanner;

public class RED
{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the max no of packets");// no of pkts to be sent
        int maxPackets= scanner.nextInt();

        System.out.println("Enter queue size"); // size of queue pkt can be stored

        int queueSize = scanner.nextInt();
        System.out.println("Enter the max probability");

        double maxProbability=scanner.nextDouble();
        System.out.println("Enter the min probability");
        double minProbability=scanner.nextDouble();

        System.out.println("Enter the threshold value");

        int threshold=scanner.nextInt();


simulateCongestion(maxPackets,queueSize,maxProbability,minProbability,threshold);


    }
    private static void simulateCongestion(int maxPackets , int queueSize, double 
maxProbability,double minProbability, int threshold)
    {
        Random rand = new Random (System.currentTimeMillis());
        int queueLength=0;

        for(int i=0; i<maxPackets;i++)
        {

            double dropProbability = 
calculateDropProbability(queueLength,queueSize,maxProbability,minProbability,thresh
old);

            if(queueLength>=threshold && rand.nextDouble() <dropProbability)
            {
                System.out.println("Packet dropped (Congestion avoidance)");
                //checking the threshold value and the probabbility to check
whether to accept or reject the packet
            }
            else{
                System.out.println("packet accepted"+(i+1));
                queueLength++;
            }
        }
    }
    private static double calculateDropProbability(int currentQueueLength, int
queueSize,double maxProbability,double minProbability,int threshold)
    {
        double slope = (maxProbability-minProbability)/(queueSize-threshold);

        return minProbability+slope*(currentQueueLength - threshold);
    }
}
```

```
Enter the max no of packets
100
Enter queue size
20
Enter the max probability
0.8
Enter the min probability
0.2
Enter the threshold value
10
packet accepted1
packet accepted2
packet accepted3
packet accepted4
packet accepted5
packet accepted6
packet accepted7
packet accepted8
packet accepted9
packet accepted10
packet accepted11
packet accepted12
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted15
packet accepted16
packet accepted17
packet accepted18
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted21
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted25
Packet dropped (Congestion avoidance)
packet accepted27
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted31
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted36
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted44
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
Packet dropped (Congestion avoidance)
packet accepted50
Packet dropped (Congestion avoidance) * 50
```

```
#Create Simulator
set ns [new Simulator]

#Open Trace file and NAM file
set ntrace [open prog1.tr w]
$ns trace-all $ntrace
set namfile [open prog1.nam w]
$ns namtrace-all $namfile

#Finish Procedure
proc Finish {} {
global ns ntrace namfile

#Dump all the trace data and close the files
$ns flush-trace
close $ntrace
close $namfile

#Execute the nam animation file
exec nam prog1.nam &

#Show the number of packets dropped
exec echo "The number of packet drops is " &
exec grep -c "^d" prog1.tr &
exit 0
}

#Create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#Label the nodes
$n0 label "TCP Source"
$n2 label "Sink"

#Set the color
$ns color 1 blue

#Create Links between nodes
#You need to modify the bandwidth to observe the variation in packet drop
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Make the Link Orientation
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right

#Set Queue Size
#You can modify the queue length as well to observe the variation in packet drop
```

```
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10

#Set up a Transport layer connection.
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0
$ns connect $tcp0 $sink0

#Set up an Application layer Traffic
set cbr0 [new Application/Traffic/CBR]
$cbr0 set type_ CBR
$cbr0 set packetSize_ 100
$cbr0 set rate_ 1Mb
$cbr0 set random_ false
$cbr0 attach-agent $tcp0
$tcp0 set class_ 1

#Schedule Events
$ns at 0.0 "$cbr0 start"
$ns at 5.0 "Finish"

#Run the Simulation
$ns run
```

```
#Create Simulator
set ns [new Simulator]

#Use colors to differentiate the traffic
$ns color 1 Blue
$ns color 2 Red

#Open trace and NAM trace file
set ntrace [open prog2.tr w]
$ns trace-all $ntrace
set namfile [open prog2.nam w]
$ns namtrace-all $namfile

#Finish Procedure
proc Finish {} {
global ns ntrace namfile

#Dump all trace data and close the file
$ns flush-trace
close $ntrace
close $namfile
#Execute the nam animation file
exec nam prog2.nam &

#Find the number of ping packets dropped
puts "The number of ping packets dropped are "
exec grep "^d" prog2.tr | cut -d " " -f 5 | grep -c "ping" &
exit 0
}

#Create six nodes
for {set i 0} {$i < 6} {incr i} {
set n($i) [$ns node]
}

#Connect the nodes
for {set j 0} {$j < 5} {incr j} {
$ns duplex-link $n($j) $n([expr ($j+1)]) 0.1Mb 10ms DropTail
}

#Define the recv function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer from $from with round trip time $rtt
ms"
}

#Create two ping agents and attach them to n(0) and n(5)
set p0 [new Agent/Ping]
$p0 set class_ 1
$ns attach-agent $n(0) $p0
set p1 [new Agent/Ping]
$p1 set class_ 1
$ns attach-agent $n(5) $p1
$ns connect $p0 $p1

#Set queue size and monitor the queue
#Queue size is set to 2 to observe the drop in ping packets
$ns queue-limit $n(2) $n(3) 2
$ns duplex-link-op $n(2) $n(3) queuePos 0.5

#Create Congestion
#Generate a Huge CBR traffic between n(2) and n(4)
set tcp0 [new Agent/TCP]
$tcp0 set class_ 2
$ns attach-agent $n(2) $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0
$ns connect $tcp0 $sink0
```

```
#Apply CBR traffic over TCP
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set rate_ 1Mb
$cbr0 attach-agent $tcp0

#Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.4 "$cbr0 start"
$ns at 0.8 "$p0 send"
$ns at 1.0 "$p1 send"
$ns at 1.2 "$cbr0 stop"
$ns at 1.4 "$p0 send"
$ns at 1.6 "$p1 send"
$ns at 1.8 "Finish"

#Run the Simulation
$ns run
```

```
#Create Simulator
set ns [new Simulator]

#Use colors to differentiate the traffics
$ns color 1 Blue
$ns color 2 Red

#Open trace and NAM trace file
set ntrace [open prog5.tr w]
$ns trace-all $ntrace
set namfile [open prog5.nam w]
$ns namtrace-all $namfile

#Use some flat file to create congestion graph windows
set winFile0 [open WinFile0 w]
set winFile1 [open WinFile1 w]

#Finish Procedure
proc Finish {} {
#Dump all trace data and Close the files
global ns ntrace namfile
$ns flush-trace
close $ntrace
close $namfile
#Execute the NAM animation file
exec nam prog5.nam &

#Plot the Congestion Window graph using xgraph
exec xgraph WinFile0 WinFile1 &
exit 0
}

#Plot Window Procedure
proc PlotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]

# To plot graph over x and y axis
puts $file "$now $cwnd"
$ns at [expr $now+$time] "PlotWindow $tcpSource $file"
}

#Create 6 nodes
for {set i 0} {$i<6} {incr i} {
set n($i) [$ns node]
}

#Create duplex links between the nodes
$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(2) $n(3) 1.0Mb 100ms DropTail

#Nodes n(3) , n(4) and n(5) are considered in a LAN
set lan [$ns newLan "$n(3) $n(4) $n(5)" 0.5Mb 40ms LL Queue/DropTail MAC/802_3
Channel]

#Orientation to the nodes
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient right

#Setup queue between n(2) and n(3) and monitor the queue
$ns queue-limit $n(2) $n(3) 20
$ns duplex-link-op $n(2) $n(3) queuePos 0.5

#Set error model on link n(2) to n(3) (optional- to analyse the amt of drop removed
pkts in tr file)
set loss_module [new ErrorModel]
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
$ns lossmodel $loss_module $n(2) $n(3)
```

```
#Set up the TCP connection between n(0) and n(4)
set tcp0 [new Agent/TCP/Newreno]
$tcp0 set fid_ 1
$tcp0 set window_ 8000
$tcp0 set packetSize_ 552
$ns attach-agent $n(0) $tcp0
set sink0 [new Agent/TCPSink/DelAck]
$ns attach-agent $n(4) $sink0
$ns connect $tcp0 $sink0



#Apply FTP Application over TCP
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP

#Set up another TCP connection between n(5) and n(1)
set tcp1 [new Agent/TCP/Newreno]
$tcp1 set fid_ 2
$tcp1 set window_ 8000
$tcp1 set packetSize_ 552
$ns attach-agent $n(5) $tcp1
set sink1 [new Agent/TCPSink/DelAck]
$ns attach-agent $n(1) $sink1
$ns connect $tcp1 $sink1

#Apply FTP application over TCP
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

#Schedule Events
$ns at 0.1 "$ftp0 start"
$ns at 0.1 "PlotWindow $tcp0 $winFile0"
$ns at 0.5 "$ftp1 start"
$ns at 0.5 "PlotWindow $tcp1 $winFile1"
$ns at 25.0 "$ftp0 stop"
$ns at 25.1 "$ftp1 stop"
$ns at 25.2 "Finish"

#Run the simulation
$ns run
```

```
#Create a ns simulator
set ns [new Simulator]

#Setup topography object
set topo [new Topography]
$topo load_flatgrid 1500 1500

#Open the NS trace file
set tracefile [open p6.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open p6.nam w]
$ns namtrace-all $namfile
$ns namtrace-all-wireless $namfile 1500 1500

#===================================
# Mobile node parameter setup
#===================================
$ns node-config -adhocRouting DSDV \
        -llType LL \
        -macType Mac/802_11 \
        -ifqType Queue/DropTail \
        -ifqLen 20 \
        -phyType Phy/WirelessPhy \
        -channelType Channel/WirelessChannel \
        -propType Propagation/TwoRayGround \
        -antType Antenna/OmniAntenna \
        -topoInstance $topo \
        -agentTrace ON \
        -routerTrace ON

#===================================
# Nodes Definition
#===================================
create-god 6
#Create 6 nodes
set n0 [$ns node]
$n0 set X_ 630
$n0 set Y_ 501
$n0 set Z_ 0.0
$ns initial_node_pos $n0 20
set n1 [$ns node]
$n1 set X_ 454
$n1 set Y_ 340
$n1 set Z_ 0.0
$ns initial_node_pos $n1 20
set n2 [$ns node]
$n2 set X_ 785
```

```
$n2 set Y_ 326
$n2 set Z_ 0.0
$ns initial_node_pos $n2 20
set n3 [$ns node]
$n3 set X_ 270
$n3 set Y_ 190
$n3 set Z_ 0.0
$ns initial_node_pos $n3 20
set n4 [$ns node]
$n4 set X_ 539
$n4 set Y_ 131
$n4 set Z_ 0.0
$ns initial_node_pos $n4 20
set n5 [$ns node]
$n5 set X_ 964
$n5 set Y_ 177
$n5 set Z_ 0.0
$ns initial_node_pos $n5 20

#===================================
# Agents Definition
#===================================
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set null1 [new Agent/Null]
$ns attach-agent $n4 $null1
$ns connect $udp0 $null1
$udp0 set packetSize_ 1500

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n3 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
$ns connect $tcp0 $sink1

#===================================
# Applications Definition
#===================================
#Setup a CBR Application over UDP connection
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 1.0Mb
$cbr0 set random_ null

#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
#===================================
# Termination
#===================================
#Define a 'finish' procedure
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
close $namfile
exec nam p6.nam &
exec echo "Number of packets dropped is : " &
exec grep -c "^D" p6.tr &
exit 0
}

$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$ftp0 start"
$ns at 180.0 "$ftp0 stop"
$ns at 200.0 "$cbr0 stop"
$ns at 200.0 "finish"
$ns at 70 "$n4 setdest 100 60 20"
$ns at 100 "$n4 setdest 700 300 20"
$ns at 150 "$n4 setdest 900 200 20"
$ns run


//===================================
// AWK file (filename.awk)
//===================================

BEGIN{
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
}
{
if($1=="r" && $3=="_1_" && $4=="RTR")
{
count1++
pack1=pack1+$8
time1=$2
}
if($1=="r" && $3=="_2_" && $4=="RTR")
{
count2++
pack2=pack2+$8
time2=$2
}
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n",
((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps \n",
((count2*pack2*8)/(time2*1000000)));
}
```