CMR Institute of Technology

Department of Information Science and Engineering

Computer Networks Lab Manual (BCS502) – Integrated

Program 1

Aim: Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.

```
set ns [new Simulator]
set tf [open lab1.tr w]
$ns trace-all $tf
set nf [open lab11.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 1000Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.001Mb 10ms DropTail
$ns set queue-limit $n0 $n2 5
$ns set queue-limit $n2 $n3 1
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp0 $null
$cbr0 set packetSize_ 5Mb
$cbr0 set interval_ 0.005
proc finish {} {
global ns nf tf
$ns flush-trace
exec nam lab11.nam &
exec echo "Number of Packets dropped :" &
exec grep -c "^d" lab11.nam &
close $tf
close $nf
exit 0
}
$ns at 0.1 "$cbr0 start"
```

```
$ns at 10.0 "finish"
$ns run
```

Program 2

Aim: Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

```
set ns [new Simulator]
set nf [open lab2.tr w]
$ns namtrace-all $nf
set tf [open lab2.nam w]
$ns namtrace-all $tf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n4 shape box
$ns duplex-link $n0 $n4 1Mb 1ms DropTail
$ns duplex-link $n1 $n4 5Mb 1ms DropTail
$ns duplex-link $n2 $n4 2Mb 1ms DropTail
$ns duplex-link $n3 $n4 2Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
```

```
$ns attach-agent $n0 $p1
$ns attach-agent $n1 $p2
$ns attach-agent $n2 $p3
$ns attach-agent $n3 $p4
$ns attach-agent $n5 $p5
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
}
$ns connect $p1 $p5
$ns connect $p3 $p4
$ns connect $p2 $p5
proc finish {} {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam lab2.nam &
exec echo "No of packets dropped" &
exec grep -c "^d" lab2.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
```

```
$ns at 0.1 "$p2 send"

$ns at 0.2 "$p2 send"

$ns at 0.1 "$p3 send"

$ns at 0.2 "$p3 send"

$ns at 0.7 "finish"

$ns run
```

Program 3

Aim: Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

```
set ns [new Simulator]

set nf [open lab3.nam w]

$ns namtrace-all $nf

set nd [open lab3.tr w]

$ns trace-all $nd

proc finish {} {

global ns nf nd

$ns flush-trace

close $nf

close $nd

exec nam lab3.nam &

exit 0

}

set n0 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
$ns duplex-link $n0 $n3 1Mb 20ms DropTail
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3
$ns duplex-link-op $n0 $n3 orient right
$ns queue-limit $n0 $n3 20
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
$tcp1 set packetSize_ 55
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_
$ns at 0.5 "$ftp1 start"
$ns at 5.0 "$ftp1 stop"
$ns at 5.5 "finish"
$ns run
```

Program 4

Aim: Develop a program for error detecting code using CRC-CCITT (16- bits).

```
import java.io.*;
class Crc {
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[ ] data;
int[ ]div;
int[ ]divisor;
int[ ]rem;
int[ ] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
```

```
rem=new int[tot_length];
crc=new int[tot_length];
/*------------------ CRC GENERATION-----------------------*/
for(int i=0;i<data.length;i++)
div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
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++) //append dividend and remainder
{

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]);

/*-------------------ERROR DETECTION---------------------*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
```

```
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
for(int j=0; j<crc.length; j++)
{
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break; }
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
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;
}
}
```

Program 5

Aim: Develop a program to implement a sliding window protocol in the data link layer.

```
import java.util.Scanner;
public class SlidingWindowProtocol {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// Input the window size
System.out.print("Enter Window Size: ");
int windowSize = scanner.nextInt();
// Input the number of frames to transmit
System.out.print("\nEnter number of frames to transmit: ");
int frameCount = scanner.nextInt();
// Input the frames
int[] frames = new int[frameCount];
System.out.println("\nEnter the " + frameCount + " frames:");
for (int i = 0; i < frameCount; i++) {
frames[i] = scanner.nextInt();
}
// Transmit frames and simulate acknowledgment
for (int i = 0; i < frameCount; i++) {
System.out.println("Frame " + frames[i] + " sent.");
// Check if window is full, then send acknowledgment
if ((i + 1) % windowSize == 0) {
```

```
System.out.println("Acknowledgment of above frames received by sender\n");
}
}
// If there are remaining frames after the last full window, send acknowledgment
if (frameCount % windowSize != 0) {
System.out.println("Acknowledgment of above frames received by sender\n");
}
scanner.close();
}
}
```

OUTPUT:

```
Enter Window Size: 3
Enter number of frames to transmit: 3
Enter the 3 frames:
25
32
13
Frame 25 sent.
Frame 32 sent.
Frame 13 sent.
Acknowledgment of above frames received by sender
Enter Window Size: 3
Enter number of frames to transmit: 7
Enter the 7 frames:
1 2 3 4 5 6 7
Frame 1 sent.
Frame 2 sent.
Frame 3 sent.
Acknowledgment of above frames received by sender
Frame 4 sent.
Frame 5 sent.
Frame 6 sent.
Acknowledgment of above frames received by sender
Frame 7 sent.
Acknowledgment of above frames received by sender
```

Program 6

Aim: Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

```
import java.util.Scanner;
public class ShortestPathAlgorithms {
// Bellman-Ford Algorithm to find the shortest path from a source vertex
public static void bellmanFord(int[][] graph, int nodesCount, int source) {
int[] dist = new int[nodesCount]; // Distance array
for (int i = 0; i < nodesCount; i++) {
dist[i] = Integer.MAX_VALUE; // Initialize distances as infinity
}
dist[source] = 0;
// Relax edges (nodesCount - 1) times
for (int i = 1; i < nodesCount; i++) {
for (int u = 0; u < nodesCount; u++) {
```

```
for (int v = 0; v < nodesCount; 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]; // Relax the edge (u, v)
}
}
}
}

// Check for negative weight cycles
for (int u = 0; u < nodesCount; u++) {
for (int v = 0; v < nodesCount; v++) {
if (graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] <
dist[v]) {
System.out.println("Graph contains negative weight cycle");
return;
}
}
}
// Output the results
System.out.println("\nBellman-Ford Algorithm (Shortest Paths from Source " + source +
"):");
for (int i = 0; i < nodesCount; i++) {
if (dist[i] == Integer.MAX_VALUE) {
System.out.println("Node " + i + ": INF");
} else {
System.out.println("Node " + i + ": " + dist[i]);
```

```
}
}
}
// Path Vector Routing Algorithm
public static void pathVectorRouting(int[][] graph, int nodesCount) {
int[][] routingTable = new int[nodesCount][nodesCount];
// Initialize routing tables with direct distances
for (int i = 0; i < nodesCount; i++) {
for (int j = 0; j < nodesCount; j++) {
if (i == j) {
routingTable[i][j] = 0; // Distance to itself is 0
} else if (graph[i][j] != 0) {
routingTable[i][j] = graph[i][j]; // Direct edge weight
} else {
routingTable[i][j] = Integer.MAX_VALUE; // No direct edge
}
}
}
// Propagate the routing table information (like distance vector)
boolean updated;
do {
updated = false;
for (int i = 0; i < nodesCount; i++) {

for (int j = 0; j < nodesCount; j++) {
if (i != j) {
// Check if a better route can be found through an intermediate node k
for (int k = 0; k < nodesCount; k++) {
if (routingTable[i][k] != Integer.MAX_VALUE && routingTable[k][j] !=
Integer.MAX_VALUE) {
int newDist = routingTable[i][k] + routingTable[k][j];
if (newDist < routingTable[i][j]) {
routingTable[i][j] = newDist;
updated = true; // Table updated
}
}
}
}
}
}
} while (updated); // Repeat until no more updates
// Display the routing table
System.out.println("\nPath Vector Routing (Final Routing Tables):");
for (int i = 0; i < nodesCount; i++) {
System.out.print("Routing Table for Node " + i + ": ");
for (int j = 0; j < nodesCount; j++) {
if (routingTable[i][j] == Integer.MAX_VALUE) {
System.out.print("INF ");
} else {
System.out.print(routingTable[i][j] + " ");
}
}
System.out.println();
}
```

```
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
// Input the number of nodes (vertices)
System.out.print("Enter the number of nodes: ");
int nodesCount = sc.nextInt();
// Create the graph as an adjacency matrix (weights of edges)
int[][] graph = new int[nodesCount][nodesCount];
System.out.print("Enter the number of edges: ");
int edgesCount = sc.nextInt();
System.out.println("Enter the edges in the format (u, v, weight):");

for (int i = 0; i < edgesCount; i++) {
int u = sc.nextInt();
int v = sc.nextInt();
int weight = sc.nextInt();
graph[u][v] = weight; // Directed edge from u to v with weight
}
// Input source node for Bellman-Ford
System.out.print("\nEnter the source node for Bellman-Ford: ");
int source = sc.nextInt();
// Call Bellman-Ford algorithm
bellmanFord(graph, nodesCount, source);
// Call Path Vector Routing algorithm
pathVectorRouting(graph, nodesCount);
sc.close();
}
```

```
}
```

Output:

```
Enter the number of nodes: 4

Enter the number of edges: 4

Enter the edges in the format (u, v, weight):

0 1 1

0 2 4

1 2 2

2 3 1

Enter the source node for Bellman-Ford: 0

Bellman-Ford Algorithm (Shortest Paths from Source 0):

Node 0: 0

Node 1: 1

Node 2: 3

Node 3: 4

Path Vector Routing (Final Routing Tables):

Routing Table for Node 0: 0 1 3 4

Routing Table for Node 1: INF 0 2 3

Routing Table for Node 2: INF INF 0 1

Routing Table for Node 3: INF INF INF 0
```

Program 7

Aim: Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

```
/*TCPClient*/

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);
System.out.println("Enter the filename");
BufferedReader keyRead=new BufferedReader(new InputStreamReader(System.in));
String fname=keyRead.readLine();
OutputStream ostream=sock.getOutputStream();
PrintWriter pwrite=new PrintWriter(ostream,true);
pwrite.println(fname);
InputStream istream=sock.getInputStream();
BufferedReader socketRead=new BufferedReader(new InputStreamReader(istream));
String str;
while((str=socketRead.readLine())!=null)
{
System.out.println(str);
}
pwrite.close();
socketRead.close();
keyRead.close();
}
}
```

TCP Server

At server side:

```
/* TCPServer */
```

```
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 ready for Connection");

Socket sock=sersock.accept();

System.out.println("Connection is Successful and waiting for chatting");

InputStream istream=sock.getInputStream();

BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));

String fname=fileRead.readLine();

BufferedReader contentRead=new BufferedReader(new FileReader(fname));

OutputStream ostream=sock.getOutputStream();


PrintWriter pwrite=new PrintWriter(ostream,true);

String str;

while((str=contentRead.readLine())!=null)

{

pwrite.println(str);

}

sock.close();

sersock.close();

pwrite.close();

fileRead.close();

contentRead.close();

}
}
```

Program 8

Aim: Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

A datagram socket is the one for sending or receiving point for a packet delivery service.

Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Source Code:

UDP Server

```
import java.io.*;
import java.net.*;
public class UDPServer
{
public static void main(String[] args)
{
DatagramSocket skt=null;
try
{
System.out.println("server is started");
skt=new DatagramSocket(6788);
byte[] buffer = new byte[1000];
while(true)
{
```

```
DatagramPacket request = new DatagramPacket(buffer,buffer.length);
skt.receive(request);
String[] message = (new String(request.getData())).split(" ");
byte[] sendMsg= (message[1].toUpperCase()+ " from server to client").getBytes();
DatagramPacket reply = new
DatagramPacket(sendMsg,sendMsg.length,request.getAddress(),request.getPort());
skt.send(reply);
}
}
catch(Exception ex)
{
System.out.println(ex.getMessage());
}
}
} UDP Client
import java.io.*;
import java.net.*;
public class UDPClient
{
public static void main(String[] args)
{
DatagramSocket skt;
try
{
skt=new DatagramSocket();
String msg= "atme college ";
```

```
byte[] b = msg.getBytes();
InetAddress host=InetAddress.getByName("127.0.0.1");
int serverSocket=6788;
DatagramPacket request =new DatagramPacket (b,b.length,host,serverSocket);
skt.send(request);
byte[] buffer =new byte[1000];
DatagramPacket reply= new DatagramPacket(buffer,buffer.length);
skt.receive(reply);
System.out.println("client received:" +new String(reply.getData()));
skt.close();
}
catch(Exception ex)
{
System.out.println(ex.getMessage());
}
}
}
```

Program 9

Aim: Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

```
import java.math.BigInteger;
public class RSA {
public static void main(String[] args) {
// Key generation
BigInteger p = BigInteger.valueOf(61);
BigInteger q = BigInteger.valueOf(53);
BigInteger n = p.multiply(q);

BigInteger phi = p.subtract(BigInteger.ONE).

BigInteger e = BigInteger.valueOf(17);

BigInteger d = e.modInverse(phi);

// Public key

System.out.println("Public Key: (" + e + ", " + n + ")");

// Private key

System.out.println("Private Key: (" + d + ", " + n + ")");

// Encryption

String message = "HELLO";

BigInteger[] ciphertext = new BigInteger[message.length()];

for (int i = 0; i < message.length(); i++) {

BigInteger m = BigInteger.valueOf(message.

ciphertext[i] = m.modPow(e, n);

}

// Decryption

String decrypted = "";

for (BigInteger c : ciphertext) {

BigInteger m = c.modPow(d, n);

decrypted += (char) m.intValue();

}

System.out.println("Decrypted: " + decrypted);

}

}
```

Program 10

Aim: Develop a program for congestion control using a leaky bucket algorithm.

```
import java.util.Scanner;
class LeakyBucketQueue {
int[] queue;
int front = 0, rear = 0;
final int maxSize = 5; // Max queue size
public LeakyBucketQueue() {
queue = new int[maxSize];
}
// Method to insert packets
void insertPackets(int packetCount) {
Scanner scanner = new Scanner(System.in);
for (int i = 0; i < packetCount; i++) {
System.out.print("Enter packet " + (i + 1) + ": ");
int packet = scanner.nextInt();
if (rear >= maxSize) {
System.out.println("Queue is full! Packet " + packet + " is lost.");
break;
} else {
queue[rear] = packet;
rear++;
}
```

```java
}}
// Method to "leak" packets (simulate processing and emptying the queue)
void leakPackets() {
if (rear == 0) {
System.out.println("Queue is empty!");
} else {
System.out.println("Leaking packets...");
for (int i = front; i < rear; i++) {
try {
Thread.sleep(1000); // Delay to simulate leaking
} catch (InterruptedException e) {
System.out.println("Error in thread sleep");
}
System.out.println("Leaked Packet: " + queue[i]);
}
// Reset the queue after leaking all packets
rear = 0;
}}
public static void main(String[] args) {
LeakyBucketQueue lbQueue = new LeakyBucketQueue();
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of packets to send: ");
int packetCount = scanner.nextInt();
lbQueue.insertPackets(packetCount); // Insert packets into the queue
lbQueue.leakPackets(); // Leak packets from the queue
}
}
```