East West Institute of Technology ,Bengaluru-91
Department of Master of Computer Applications

LAB MANUAL

SCHEME: 2022

COURSE: COMPUTER NETWORKS LAB

CODE: 22MCAL17

SEMESTER : 01

ACADEMIC YEAR – 2022-23

COMPUTER NETWORK LABORATORY (Effective from the academic year 2021 -2022) SEMESTER – I			
Subject Code	22MCAL17	**CIE Marks**	50
Number of Contact Hours/Week	0:3:0	**SEE Marks**	50
Credits	1.5	**Exam Hours**	3 Hrs

Course objectives:

- To understand the working principle of various communication protocols.
- To understand the network simulator environment and visualize a network topology and observe its performance

Experiments

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

2. Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.
3.Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP
4. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.
5. Implement Dijsktra's algorithm to compute the shortest path through a network
6. Implement data encryption and data decryption
7. Simulate the network with five nodes nO, nl, n2, n3, n4, forming a star topology. The node n4 is at the centre. Node nO is a TCP source, which transmits packets to node n3 (a TCP sink) through the node n4. Node nl is another traffic source, and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds.
8. Simulate to study transmission of packets over Ethernet LAN and determine the number of packets drop destination.
Demonstration Experiments (For CIE) if any
9. Simulate the different types of internet traffic such as FTP and TELNET over a wired network and analyze the packet drop and packet delivery ratio in the network
Course outcomes (Course Skill Set): **At the** end of the course the student will be able to: • Implement data link layer farming methods. • Analyze error detection and error correction codes. • Implement and analyze routing and congestion issues in network design. • Implement Encoding and Decoding techniques used in presentation layer. To be able to work with different network tools

Contents

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Implement data link layer farming methods.
- Analyze error detection and error correction codes.
- Implement and analyze routing and congestion issues in network design.
- Implement Encoding and Decoding techniques used in presentation layer.

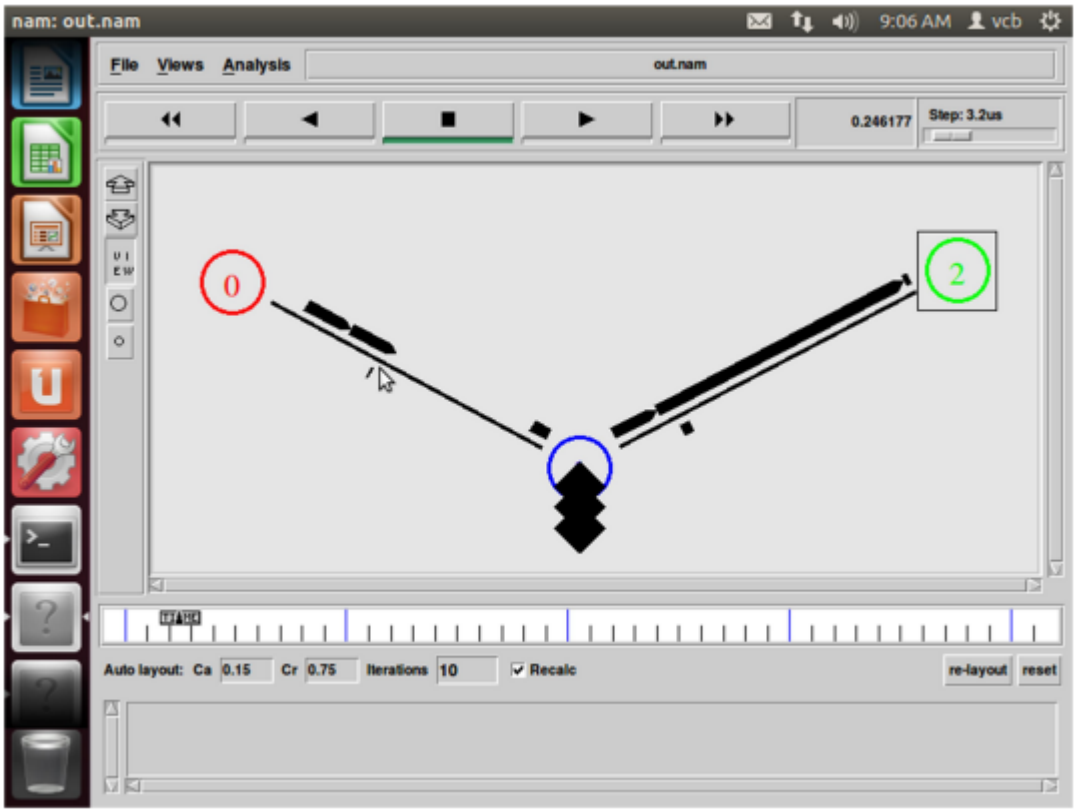
To be able to work with different network tools

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

```
#Create simulator
set ns [new Simulator]
set tracefile [open lab1.tr w]
$ns trace-all $tracefile
set namfile [open lab1.nam w]
$ns namtrace-all $namfile
#create node
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
#create link
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.5Mb 10ms DropTail
#Set Queue Size
$ns queue-limit $n0 $n1 05
$ns queue-limit $n1 $n2 05
#setup tcp connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
#set sink to node
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
#connect tcp src and sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set packetSize_ 1024
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
```

```
close $namfile

set awkCode {
BEGIN {}
{
if ($1 == "d" && $4 = 2 ) {
dcount = dcount + $6
print $2, dcount >> "lab1.data"
}
}
END {}
}
exec awk $awkCode lab1.tr
exec xgraph -bb -tk -x Time -y Bytes lab1.data -bg white &
exec nam lab1.nam &
exit 0
}
$ns at 0.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 5.0 "finish"
$ns run
```



2. Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.

```
#include <stdio.h>
int main() {
   int i, j, count = 0;
   int data[20], stuffed_data[30];
   // Read the input data
   printf("Enter the data: ");
```

```
    for(i = 0; i < 20; i++) {
        scanf("%d", &data[i]);
    }
    // Perform bit stuffing
    j = 0;
    for(i = 0; i < 20; i++) {
        stuffed_data[j] = data[i];
        if(data[i] == 1) {
            count++;
        }
        else {
            count = 0;
        }
        j++;
        if(count == 5) {
            stuffed_data[j] = 0;
            j++;
            count = 0;
        }
    }

    // Print the stuffed data
    printf("Stuffed data: ");
    for(i = 0; i < j; i++) {
        printf("%d ", stuffed_data[i]);
    }
    printf("\n");

    return 0;
}
```

This program takes an input string from the user and performs character stuffing using start and end delimiters and an escape character. The start and end delimiters are defined as !, and the escape character is defined as \. The input string is scanned using scanf and then the stuffed string is created using a for loop. Each character in the input string is checked to see if it matches the start or end delimiter or the escape character. If it does, an escape character is added to the stuffed string before the character. Finally, the start and end delimiters are added to the beginning and end of the stuffed string, respectively, and the stuffed string is printed to the console.

```
#include <stdio.h>
#include <string.h>
int main() {
    char start_delimiter = '!';
    char end_delimiter = '!';
    char escape_character = '\\';
    char input_string[100], stuffed_string[100];
    int i, j = 0;
    printf("Enter the input string: ");
    scanf("%s", input_string);
    stuffed_string[j++] = start_delimiter;
    for (i = 0; i < strlen(input_string); i++) {
        if (input_string[i] == start_delimiter || input_string[i] == end_delimiter || input_string[i] == escape_character) {
            stuffed_string[j++] = escape_character;
        }
        stuffed_string[j++] = input_string[i];
    }
    stuffed_string[j++] = end_delimiter;
    stuffed_string[j] = '\0';
    printf("The stuffed string is: %s", stuffed_string);
    return 0;
}
```

Output:

```
Enter the input string: hello!world!
The stuffed string is: !hello\!world\!!
```

3.Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP

The Third Program is already shared in Google Classroom.

https://classroom.google.com/c/NTE3Mzk3NjkwNTc0/m/NjA0NzMyNTM2MzIy/details

4. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.

```
#include<stdio.h>
int main()
{
	int windowsize,sent=0,ack,i;
	printf("enter window size\n");
	scanf("%d",&windowsize);
	while(1)
	{
		for( i = 0; i < windowsize; i++)
			{
				printf("Frame %d has been transmitted.\n",sent);
				sent++;
				if(sent == windowsize)
					break;
			}
			printf("\nPlease enter the last Acknowledgement received.\n");
			scanf("%d",&ack);

			if(ack == windowsize)
				break;
			else
				sent = ack;
	}
return 0;
}
```

output:-

```
enter window size
8
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
```

```
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
8
```

5. Implement Dijsktra's algorithm to compute the shortest path through a network

```
#include <stdio.h>
#include <stdlib.h>
int cost[10][10],dist[10],s;
dijkstra(int n)
{
int count=1,u,i,v,flag[10]={0},min;
for(i=1;i<=n;i++)
{

dist[i]=999;
}
dist[s]=0;
while(count<=n)
{
min=99;
for(v=1;v<=n;v++)
if(dist[v]<min && !flag[v])
min=dist[v],u=v;
flag[u]=1;
count++;
for(v=1;v<=n;v++)
if((dist[u]+cost[u][v]< dist[v])&& !flag[v])
dist[v]=dist[u]+cost[u][v];
}
}
int main()
{
int i,j,n;
printf("Enter the vertices:");
scanf("%d",&n);
printf("\n enter the cost matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
```

```
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0) cost[i][j]=999;
}
printf("\n enter the source vertex:");
scanf("%d",&s);
dijkstra(n);
printf("\n the shortest path \n");
for(i=1;i<=n;i++)
{
if(i!=s)
printf(" %d -> %d , cost =%d\n",s,i,dist[i]);
}
return 0;
}
```

Output:

```
 Enter the no of routers:4

 Enter the distance matrix values
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

 Distance Vector Matrix
  0 10  3  4
  2  0  5  6
  7  7  0  1
  6 16  9  0
```

6. Implement data encryption and data decryption

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Algorithm

Implementation of RSA Algorithm

1. Generate two large random primes, P and Q, of approximately equal size
2. Compute $N = P \times Q$
3. Compute $Z = (P-1) \times (Q-1)$.
4. Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$
5. Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$
6. The public key is (N, E) and the private key is (N, D).

Note: The values of P, Q, and Z should also be kept secret.

The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes P=11, Q=3.
2. $N = P \times Q = 11 \times 3 = 33$

$Z = (P-1) \times (Q-1) = 10 \times 2 = 20$

3. Lets choose E=3

Check GCD(*E*, *P*-1) = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
and check GCD(*E*, *Q*-1) = GCD(3, 2) = 1
therefore GCD(*E*, *Z*) = GCD(3, 20) = 1
4. Compute *D* such that *E* x *D* ≡ 1 (mod *Z*)
compute *D* = *E*^-1 mod *Z* = 3^-1 mod 20
find a value for *D* such that *Z* divides ((*E* x *D*)-1)
find *D* such that 20 divides 3D-1.
Simple testing (D = 1, 2, ...) gives D = 7
Check: (*E* x *D*)-1 = 3.7 - 1 = 20, which is divisible by *Z*.
5. Public key = (*N*, *E*) = (33, 3)
Private key = (*N*, *D*) = (33, 7).
Now say we want to encrypt the message m = 7,
Cipher code = *M*^*E* mod *N*
= 7^3 mod 33
= 343 mod 33
= 13.
Hence the ciphertext c = 13.
To check decryption we compute Message' = *C*^*D* mod *N*
= 13^7 mod 33 = 7.
Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

Program

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <math.h>
int mult(unsigned int x, unsigned int y, unsigned int n) {
unsigned long int k=1;
int j;
for (j=1; j<=y; j++) k = (k * x) % n;
return (unsigned int) k;
}
void main () {
char msg[100];
unsigned int pt[100], ct[100], n, d, e, p, q, i;
printf("Enter message : "); gets(msg);
//strcpy(pt, msg);
for(i=0;i<strlen(msg);i++)
pt[i]=msg[i];
n = 253; d = 17; e = 13;
printf("\nCT = ");
for(i=0; i<strlen(msg); i++) ct[i] = mult(pt[i], e,n);
for(i=0; i<strlen(msg); i++) printf("%d ", ct[i]);
printf("\nPT = ");
for(i=0; i<strlen(msg); i++) printf("%c", pt[i]);
```

```
for(i=0; i<strlen(msg); i++) pt[i] = mult(ct[i], d,n) ;
}
```

Output

Enter message : alpha

CT = 113 3 129 213 113

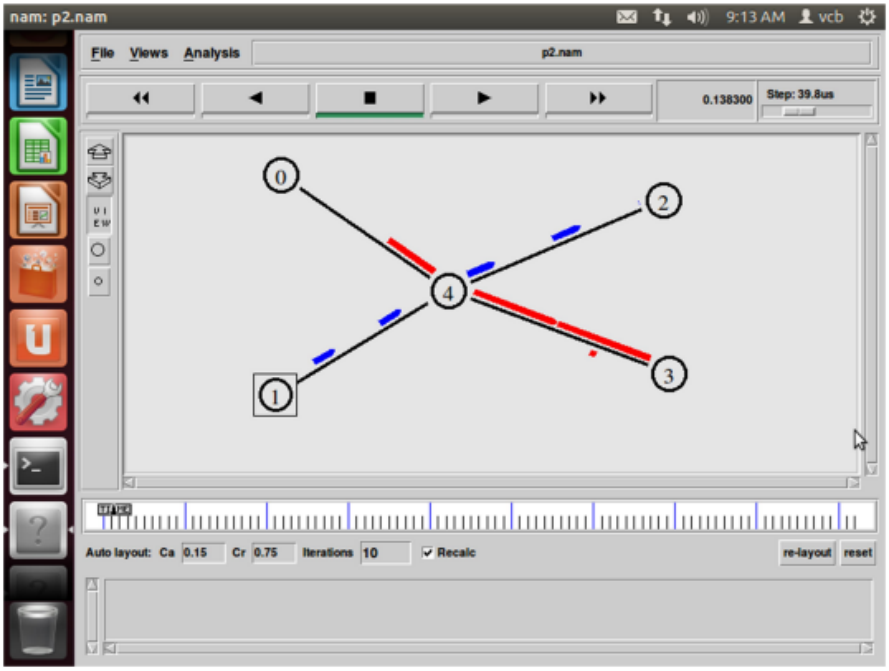
PT = alpha

7. Simulate the network with five nodes nO, nl, n2, n3, n4, forming a star topology. The node n4 is at the centre. Node nO is a TCP source, which transmits packets to node n3 (a TCP sink) through the node n4. Node nl is another traffic source, and sends UDP packets to node n2 through n4. The duration of the simulation time is 10 seconds.

```
set ns [new Simulator]
set namfile [open ex_01.nam w]
$ns namtrace-all $namfile
set tracefile [open ex_01.tr w]
$ns trace-all $tracefile
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n1 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n3 1Mb 10ms DropTail
$ns duplex-link $n4 $n2 1Mb 10ms DropTail
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n2 $null
$ns connect $udp $null
$udp set class_ 1
$ns color 1 Blue
$tcp set class_ 2
$ns color 2 Red
```

```
set cbr [new Application/Traffic/CBR]
$cbr set packetsize_ 500
$cbr set interval_ 0.005
$cbr attach-agent $udp
$ns at 0.0 "$cbr start"
$ns at 0.0 "$ftp start"
$ns at 9.0 "$cbr stop"
$ns at 9.0 "$ftp stop"
proc finish {} {
global ns namfile tracefile
$ns flush-trace
close $namfile
close $tracefile
exec nam ex_01.nam &
exit 0
}
$ns at 10.0 "finish"
$ns run
```

Output:



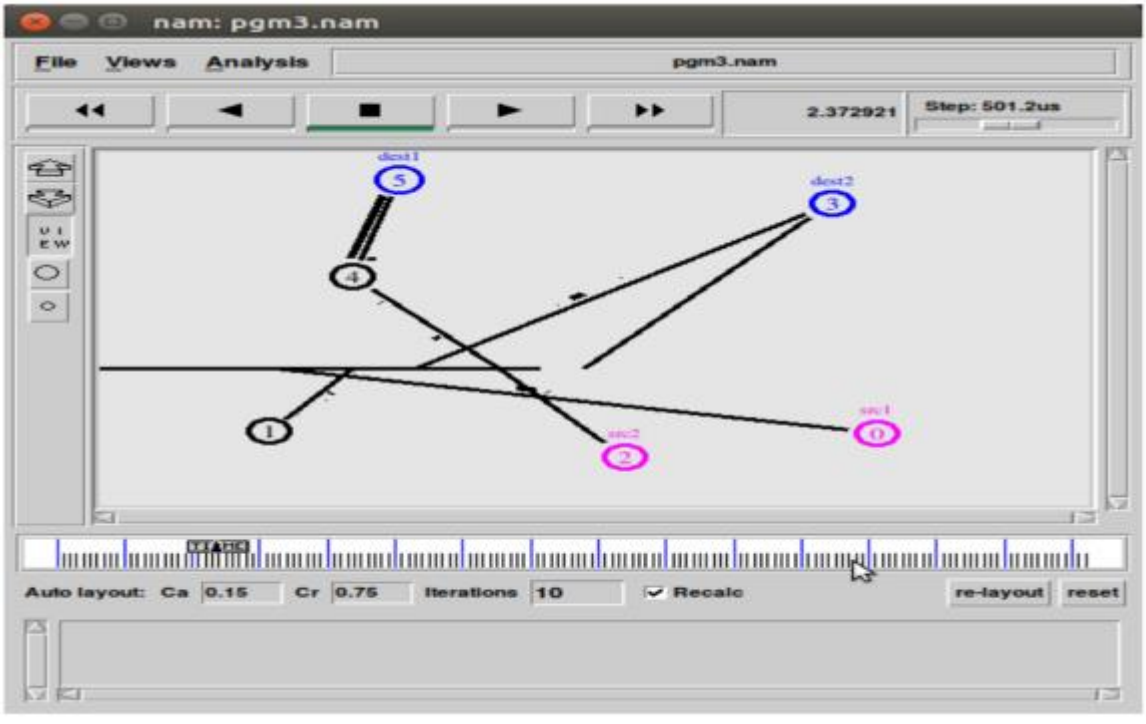
8. Simulate to study transmission of packets over Ethernet LAN and determine the number of packets drop destination.

```
set ns [new Simulator]
set tf [open pgm3.tr w]
$ns trace-all $tf
```

```
set nf [open pgm3.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
$n0 label "src1"
set n1 [$ns node]
set n2 [$ns node]
$n2 label "src2"
set n3 [$ns node]
$n3 label "dest2"
set n4 [$ns node]
set n5 [$ns node]
$n5 label "dest1"
$n0 color "magenta"
$n2 color "magenta"
$n5 color "blue"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail
Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3
set file1 [open file1.tr w]
$tcp0 attach $file1
```

```
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp0 trace cwnd_
$tcp2 trace cwnd_
proc finish { } {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam pgm3.nam &
exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

Output:



Demonstration Experiments (For CIE) if any

9. Simulate the different types of internet traffic such as FTP and TELNET over a wired network and analyze the packet drop and packet delivery ratio in the network

```
set ns [new Simulator]
set ntrace [open prog.tr w]
$ns trace-all $ntrace
set namfile [open prog5.nam w]
$ns namtrace-all $namfile
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 1Mb 10ms DropTail
$ns simplex-link $n3 $n2 1Mb 10ms DropTail
$ns queue-limit $n0 $n2 10
$ns simplex-link-op $n0 $n2 queuePos 0.5
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0
set telnet [new Application/Telnet]
$telnet attach-agent $udp0
$telnet set type_ Telnet
proc Finish {} {
global ns ntrace namfile
$ns flush-trace
close $ntrace
close $namfile
```

```
exec nam prog5.nam &
set numTcp [exec grep "^r" prog5.tr | grep "tcp" | tail -n 1 | cut
-d " " -f 6]
set TcpSize [exec grep "^r" prog5.tr | grep -c "tcp"]
set tcpTime 24.0
set numUdp [exec grep "^r" prog5.tr | grep "udp" | tail -n 1 | cut
-d " " -f 6]
set UdpSize [exec grep "^r" prog5.tr | grep -c "udp"]
set UdpTime 23.9
puts "The throughput of FTP is"
puts "[expr ($numTcp*$TcpSize)/$tcpTime] bytes per second"
puts "The throughput of Telnet is"
puts "[expr ($numUdp*$UdpSize)/$UdpTime] bytes per second"
exit 0
}
$ns at 0.1 "$telnet start"
$ns at 0.5 "$ftp0 start"
$ns at 23.9 "$telnet stop"
$ns at 24.0 "$ftp0 stop"
$ns at 25.0 "Finish"
$ns run
```

OUTPUT

```
The throughput of FTP is
243706.66666666666 bytes per second
The throughput of Telnet is
1422.5941422594144 bytes per second
```