Part A

1. Client – Server

Server.c

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
    char fname[50], buffer[1025];
    int req, res, n, file;
    mkfifo("req.fifo", 0777);
    mkfifo("res.fifo", 0777);
    printf("Waiting for request...\n");
    req = open("req.fifo", O_RDONLY);
    res = open("res.fifo", O_WRONLY);
    read(req, fname, sizeof(fname));
    printf("Received request for %s\n", fname);
    file = open(fname, O_RDONLY);
    if (file < 0)
        write(res, "File not found\n", 15);
    else {
        while((n = read(file, buffer, sizeof(buffer))) > 0) {
            write(res, buffer, n);
        }
    }
    close(req);
    close(res);
    unlink("req.fifo");
    unlink("res.fifo");
    return 0;
}
```

Client.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
    char fname[50], buffer[1025];
    int req, res, n;
    req = open("req.fifo", O_WRONLY);
    res = open("res.fifo", O_RDONLY);
    if(req < 0 || res < 0) {
        printf("Please Start the server first\n");
        exit(-1);
    }
    printf("Enter filename to request:\n");
    scanf("%s", fname);
    write(req, fname, sizeof(fname));
    printf("Received response\n");
    while((n = read(res, buffer, sizeof(buffer)))>0) {
        write(1, buffer, n);
    }
    close(req);
    close(res);
    return 0;
}
```

2. Traffic Management implementing Leaky Bucket Algorithm

```
#include<stdio.h>
#include<stdlib.h>
#define MIN(x,y) (x>y)?y:x

int main()
{
      int orate,drop=0,cap,x,count=0,
      inp[10]={0},i=0,nsec,ch;
      printf(" \n enter bucket size : ");
      scanf("%d",&cap);
      printf("\n enter output rate : ");
      scanf("%d",&orate);
      do
      {
            printf("\n enter number of packets coming at second %d : ",i+1);
            scanf("%d",&inp[i]);
            i++;
            printf("\n enter 1 to contiue or 0 to quit.........");
            scanf("%d",&ch);
      }while(ch);
      nsec=i;
      printf("\nsecond \trecieved \tsent \tdropped \tremained \n");
      for(i=0;count || i<nsec;i++)
      {
            printf("%d",i+1);
            printf(" \t %d\t ",inp[i]);
            printf(" \t %d\t ",MIN((inp[i]+count),orate));
            if((x=inp[i]+count-orate)>0)
            {
                  if(x>cap)
                  {
                        count=cap;
                        drop=x-cap;
                  }
                  else
                  {
                        count=x;
                        drop=0;
                  }
            }
            else
            {
                  drop=0;
                  count=0;
            }
            printf(" \t %d \t %d \n",drop,count);
      }
      return 0;
}
```

3. Bellman Ford Algorithm

```
#include<stdio.h>
struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];
int main()
{
    int costmat[20][20];
    int nodes,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
    scanf("%d",&nodes);
    printf("\nEnter the cost matrix :\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            scanf("%d",&costmat[i][j]);
            costmat[i][i]=0;
            rt[i].dist[j]=costmat[i][j];
            rt[i].from[j]=j;
        }
    }
        do
        {
            count=0;
            for(i=0;i<nodes;i++)
            for(j=0;j<nodes;j++)
            for(k=0;k<nodes;k++)
                if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
                {//We calculate the minimum distance
                    rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
                    rt[i].from[j]=k;
                    count++;
                }
        }while(count!=0);
        for(i=0;i<nodes;i++)
        {
            printf("\n\n For router %d\n",i+1);
            for(j=0;j<nodes;j++)
            {
                printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]
+1,rt[i].dist[j]);
            }
        }
    printf("\n\n");
}
```

4. Dijkstra’s

```
#include<stdio.h>
#include<stdlib.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;
    printf("Enter no. of vertices : ");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix :\n");

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);

    printf("\nEnter the starting node : ");
    scanf("%d",&u);
    dijkstra(G,n,u);

    return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;
            else
                cost[i][j]=G[i][j];

    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
    }

    distance[startnode]=0;
    visited[startnode]=1;
    count=1;
```

```
    while(count<n-1)
    {
        mindistance=INFINITY;

        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }

            visited[nextnode]=1;
            for(i=0;i<n;i++)
                if(!visited[i])
                    if(mindistance+cost[nextnode][i]<distance[i])
                    {
                        distance[i]=mindistance+cost[nextnode][i];
                        pred[i]=nextnode;
                    }
        count++;
    }

    for(i=0;i<n;i++)
        if(i!=startnode)
        {
            printf("\nDistance of node %d = %d ",i,distance[i]);
            printf("\nPath = %d",i);

            j=i;
            do
            {
                j=pred[j];
                printf(" <- %d",j);
            }while(j!=startnode);
    }
}
```

5. CRC Technique

```
#include<stdio.h>
#include<string.h>
#define N strlen(g)

char t[28],cs[28],g[]="10001000000100001";
int a,e,c;

void xor(){
    for(c = 1;c < N; c++)
    cs[c] = (( cs[c] == g[c])?'0':'1');
}

void crc(){
    for(e=0;e<N;e++)
        cs[e]=t[e];
    do{
        if(cs[0]=='1')
            xor();
        for(c=0;c<N-1;c++)
            cs[c]=cs[c+1];
        cs[c]=t[e++];
    }while(e<=a+N-1);
}

int main()
{
    printf("\nEnter data : ");
    scanf("%s",t);
    printf("\nGeneratng polynomial : %s",g);
    a=strlen(t);
    for(e=a;e<a+N-1;e++)
        t[e]='0';
    printf("\nModified data is : %s",t);
    crc();
    printf("\nChecksum is : %s",cs);
    for(e=a;e<a+N-1;e++)
        t[e]=cs[e-a];
    printf("\nFinal codeword is : %s",t);
    printf("\nTest error detection 0(yes) 1(no)? : ");
    scanf("%d",&e);
    if(e==0)
    {
        do{
            printf("\nEnter the position where error is to be inserted : ");
            scanf("%d",&e);
        }while(e==0 || e>a+N-1);
        t[e-1]=(t[e-1]=='0')?'1':'0';
        printf("\nErroneous data : %s\n",t);
    }
    crc();
    for(e=0;(e<N-1) && (cs[e]!='1');e++);
        if(e<N-1)
            printf("\nError detected\n\n");
        else
            printf("\nNo error detected\n\n");
        return 0;
}
```

6. Internet Checksum for error correction & detection

```
#include<stdio.h>
#include<string.h>

//for encoding
int checksum1(int fl)
{
char in[100];
int buf[25];
int i,sum1=0,n,temp,temp1;
scanf("%s",in);
if(strlen(in)%2!=0)
        n=(strlen(in)+1)/2;
else
        n=n=(strlen(in))/2;
for(i=0;i<n;i++)
    {
temp=in[i*2];
temp=(temp*256)+in[(i*2)+1];
sum1=sum1+temp;
    }

if(sum1%65536!=0)
    {
        n=sum1%65536;
sum1=(sum1/65536) + n;
    }
sum1=65535-sum1;
printf("%x\n",sum1);
return sum1;
}

//for decoding
int checksum2(int fl)
{
char in[100];
int buf[25];
int i,sum2=0,n,temp,temp1;
scanf("%s",in);
if(strlen(in)%2!=0)
        n=(strlen(in)+1)/2;
else
        n=n=(strlen(in))/2;
for(i=0;i<n;i++)
    {
temp=in[i*2];
temp=(temp*256)+in[(i*2)+1];
sum2=sum2+temp;
    }

if(sum2%65536!=0)
    {
        n=sum2%65536;
sum2=(sum2/65536) + n;
    }
sum2=65535-sum2;
return sum2;
}
```

```
void main()
{
int ch,sum1,sum2;
do{
printf("1.Encode \n2.Decode \n3.Exit \n");
scanf("%d",&ch);
switch(ch)
        {

case 1: printf("Enter the string \n");
sum1=checksum1(0);
printf("Checksum to append is:%x \n",sum1);
break;
case 2: printf("Enter the string \n");
sum2=checksum2(1);
      if(sum1!=sum2){
            printf("The data has been tampered with or invalid checksum\n");
            printf("The syndrome bit is %x\n",sum2);
      }
      else
            printf("The checksum is valid %x \n",sum2);
break;
case 3: break;
default: printf("Invalid option, try again \n");
        }
    }
while(ch!=3);
}
```

7. Priority Queue

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();

int pri_que[MAX];
int front, rear;

void main()
{
    int n, ch;
    printf("\n1 - Insert an element into queue");
    printf("\n2 - Delete an element from queue");
    printf("\n3 - Display queue elements");
    printf("\n4 - Exit");
    create();
    while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter value to be inserted : ");
            scanf("%d",&n);
            insert_by_priority(n);
            break;
        case 2:
            printf("\nEnter value to delete : ");
            scanf("%d",&n);
            delete_by_priority(n);
            break;
        case 3:
            display_pqueue();
            break;
        case 4:
            exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}
```

```
void create()
{
      front = rear = -1;
}

void insert_by_priority(int data)
{
      if (rear >= MAX - 1)
      {
            printf("\nQueue overflow no more elements can be inserted");
            return;
      }

      if ((front == -1) && (rear == -1))
      {
            front++;
            rear++;
            pri_que[rear] = data;
            return;
      }
      else
            check(data);
      rear++;
}

void delete_by_priority(int data)
{
      int i;
      if ((front==-1) && (rear==-1))
      {
            printf("\nQueue is empty no elements to delete");
            return;
      }

      for (i = 0; i <= rear; i++)
      {
            if (data == pri_que[i])
            {
                  for (; i < rear; i++)
                  {
                        pri_que[i] = pri_que[i + 1];
                  }
                  pri_que[i] = -99;
                  rear--;
            if (rear == -1)
            front = -1;
            return;
            }
      }
      printf("\n%d not found in queue to delete", data);
}
```

```
void check(int data)
{
      int i,j;
      for (i = 0; i <= rear; i++)
      {
            if (data >= pri_que[i])
            {
                  for (j = rear + 1; j > i; j--)
                  {
                        pri_que[j] = pri_que[j - 1];
                  }
                  pri_que[i] = data;
                  return;
            }
      }
      pri_que[i] = data;
}

void display_pqueue()
{
      if ((front == -1) && (rear == -1))
      {
            printf("\nQueue is empty");
            return;
      }
      for (; front <= rear; front++)
      {
            printf(" %d ", pri_que[front]);
      }
      front = 0;
}
```

Part B

1. Three nodes point-to-point network

p1.tcl

```
set ns [new Simulator]
set nf [open p1.nam w]
$ns namtrace-all $nf
set nd [open p1.tr w]
$ns trace-all $nd

proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec awk -f p1.awk p1.tr &
exec nam p1.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 512Kb 10ms DropTail
$ns queue-limit $n1 $n2 5

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $udp0 $sink

$ns at 0.2 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

p1.awk

```
BEGIN {
dcount = 0;
rcount = 0;
}
{
event = $1;
if(event == "d")
      dcount++;
if(event == "r")
      rcount++;
}
END {
printf("The no.of packets dropped  : %d\n ",dcount);
printf("The no.of packets recieved : %d\n ",rcount);
}
```

2. Internet traffic such as FTP and TELNET

p2.tcl

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

#Open Trace and NAM Trace File
set ntrace [open p2.tr w]
$ns trace-all $ntrace
set namfile [open p2.nam w]
$ns namtrace-all $namfile

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

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

#Execute the nam animation file
exec awk -f p2.awk p2.tr &
exec nam p2.nam &
exit 0
}

$ns color 1 Blue
$ns color 2 Red

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$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

#Set queue size and Monitor the queue
$ns queue-limit $n0 $n2 10
$ns simplex-link-op $n0 $n2 queuePos 0.5

#Set TCP Connection between n0 and n3
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
$tcp0 set fid_ 1

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

```
#Set TCP Connection between n1 and n3
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
$tcp1 set fid_ 2

#Attach Telnet Application over UDP
set telnet [new Application/Telnet]
$telnet attach-agent $tcp1
$telnet set type_ Telnet

#Schedule Events
$ns at 0.5 "$telnet start"
$ns at 0.5 "$ftp0 start"
$ns at 24.5 "$telnet stop"
$ns at 24.5 "$ftp0 stop"
$ns at 25.0 "Finish"
$ns run
```

p2.awk

```
BEGIN {
numTCP1=0;
tcpSize1=0;
numTCP2=0;
tcpSize2=0;
totaltcp1=0;
totaltcp2=0;
}
{
event=$1;
pkttype= $5;
fromnode=$9;
tonode=$10;
pktsize=$6;
if(event == "r" && pkttype == "tcp" && fromnode == "0.0" && tonode == "3.0")
{
numTCP1++;
tcpSize1 = pktsize;
}
if(event == "r" && pkttype == "tcp" && fromnode == "1.0" && tonode == "3.1")
{
numTCP2++;
tcpSize2 = pktsize;
}
}
END {
totaltcp1=numTCP1*tcpSize1*8;
totaltcp2=numTCP2*tcpSize2*8;
throughputtcp1= totaltcp1/24;
throughputtcp2= totaltcp2/24;
printf("The Throughput of FTP application is %d \n", throughputtcp1);
printf("The Throughput of TELNET application is %d \n", throughputtcp2);
}
```

3. Four node point-to-point network

p3.tcl

```
set ns [new Simulator]
set nf [open p3.nam w]
$ns namtrace-all $nf
set nd [open p3.tr w]
$ns trace-all $nd

proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec awk -f p3.awk p3.tr &
exec nam p3.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$n1 color blue
$n0 color red
$n2 color purple
$n3 color orange

$ns color 1 blue

$n0 label TCP
$n1 label UDP
$n3 label NULL-TCPSINK

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
$tcp0 set fid_ 1

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

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 cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

```
$ns at 0.2 "$cbr0 start"
$ns at 0.1 "$ftp0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 4.4 "$ftp0 stop"

$ns at 5.0 "finish"
$ns run
```

p3.awk

```
BEGIN {
ctcp=0;
cudp=0;
}
{
pkt=$5;
if(pkt=="cbr") { cudp++;}
if(pkt=="tcp") { ctcp++;}
}
END {
printf("No of packets sent\nTcp : %d\nUdp : %d\n",ctcp,cudp);
}
```

4. Ping messages over a 6 node network

p4.tcl

```
set ns [new Simulator]
set nf [open p4.nam w]
$ns namtrace-all $nf
set nd [open p4.tr w]
$ns trace-all $nd

proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec awk -f p4.awk p4.tr &
exec nam p4.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$ns duplex-link $n1 $n0 1Mb 12ms DropTail
$ns duplex-link $n2 $n0 1Mb 10ms DropTail
$ns duplex-link $n3 $n0 1Mb 10ms DropTail
$ns duplex-link $n4 $n0 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail
$ns duplex-link $n6 $n0 1Mb 11ms DropTail

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] recieved ping answer from \
$from with round-trip-time $rtt ms."
}

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]
set p6 [new Agent/Ping]

$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6

$ns queue-limit $n0 $n4 3
$ns queue-limit $n0 $n5 2
$ns queue-limit $n0 $n6 2
```

```
$ns connect $p1 $p4
$ns connect $p2 $p5
$ns connect $p3 $p6

$ns at 0.1 "$p1 send"
$ns at 0.3 "$p2 send"
$ns at 0.5 "$p3 send"
$ns at 1.0 "$p4 send"
$ns at 1.2 "$p5 send"
$ns at 1.4 "$p6 send"
$ns at 2.0 "finish"
$ns run
```

p4.awk

```
BEGIN {count=0;
}
{
event=$1;
if(event=="d")
{
count++;
}
}
END {
printf("No of packets dropped : %d\n",count);
}
```

5. Ethernet Lan to determine collision across different nodes

p5.tcl

```
#Lan simulation - mac.tcl
set ns [new Simulator]

#define color for data flows
$ns color 1 Blue
$ns color 2 Red

#open tracefile
set tracefile1 [open p5.tr w]
$ns trace-all $tracefile1

#open nam file
set namfile [open p5.nam w]
$ns namtrace-all $namfile

#define the finish procedure
proc finish {} {
      global ns tracefile1 namfile
      $ns flush-trace
      close $tracefile1
      close $namfile
      exec awk -f p5.awk p5.tr &
      exec nam p5.nam &
      exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Specify color and shape for nodes
$n1 color Red
$n1 shape box
$n5 color Red
$n5 shape box
$n0 color Blue
$n4 color Blue

#create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

# Create a LAN
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel]

#Give node position
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left
```

```
#setup TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packet_size_ 552

#set ftp over tcp connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2

#setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.05Mb
$cbr set random_ false

#scheduling the events
$ns at 0.0 "$n0 label TCP_Traffic"
$ns at 0.0 "$n1 label UDP_Traffic"
$ns at 0.3 "$cbr start"
$ns at 0.8 "$ftp start"
$ns at 7.0 "$ftp stop"
$ns at 7.5 "$cbr stop"
$ns at 8.0 "finish"
$ns run
```

p5.awk

```
BEGIN {
pktdrp=0;
}
{
event=$1;
if(event == "d") {
pktdrp++; }
}
END {
printf("The number of packets dropped is %d\n",pktdrp);
}
```

6. Ethernet LAN to compare throughput

p6.tcl

```
set ns [new Simulator]

set nf [open p6.nam w]
$ns namtrace-all $nf

set nd [open p6.tr w]
$ns trace-all $nd

proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec awk -f p6.awk p6.tr &
exec nam p6.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6" 0.2Mb 40ms LL Queue/DropTail
Mac/802_3

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 5.5 "finish"
$ns run
```

p6.awk

```
BEGIN {
sSize=0;
startTime = 5.0;
stopTime = 0.1;
Tput = 0;
}
{
event = $1;
time = $2;
from = $3;
to = $4;
pkt = $5;
size = $6;
fid = $7;
src = $8;
dst = $9;
seqn = $10;
pid = $11;
if (event == "+") {
if(time < startTime) {
startTime = time;
}}
if (event == "r") {
if(time > stopTime) {
stopTime = time;
}
sSize+=size;
}
Tput = (sSize/(stopTime-startTime))*(8/1000);
printf("%f\t%.2f\n",time,Tput);
}
END {
}
```