NETWORK LAB – CS2307

YEAR: III **SEMESTER: VI**

BRANCH\SEC: IT-A

LIST OF PROGRAMS

1. Write a program to simulate ARP/ RARP protocol
2. Write a program to perform bit stuffing on the sender side
3. Write a program to remove the extra bits from the received bits
4. Write a program to compute Cyclic redundancy check
5. Write a program to simulate the sliding window protocol for Go back n
6. Write a program to simulate the sliding window protocol for Selective reject
7. Write a program to simulate the Stop and Wait protocol
8. Write a program to download a file using RS232 interface
9. Write a program for client Server chat application
10. Write a program to find the port address for service name
11. Write a program to simulate OSPF protocol
12. Study of NS2
13. Study of Glomosim/OPNET

1. SIMULATION OF ARP / RARP PROTOCOL

AIM:

To write a program to find the Physical Address for a given IP address using Simulation.

ALGORITHM:

```
Step 1: Start the program
Step 2: Generate random numbers to get IP addresses and physical addresses
Step 3: Store the IP address and physical addresses in an array
Step 4: Randomly choose an IP address and identify the corresponding Physical address
for that IP address with the help of the array.
Step 5: If a match is found then display the IP address and the corresponding physical
address otherwise display that no match was found.
Step 6: Similarly choose a physical address randomly and find the corresponding IP
address for that Physical address with the help of the array.
Step 7: If a match is found then display the physical address and the corresponding IP
address otherwise display that no match was found.
Step 8: Stop the program.
```

CODING:

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
void main()
{
struct
{
char *ipa;
char *pha;
}ipadd[2];
int i,j,x=10;
char *temp,*temp1,*x1;
char ip[2][15]={"135.237.105.128","225.22.205.221"};
char ph[2][18]={"SYSTEM1","SYSTEM12"};
clrscr();
for(j=0;j<2;j++)
{
srand(x++);
for(i=0;i<4;i++)
{
itoa(rand()%256,temp,10);
printf("\n%s",temp);
strcat(ipadd[j].ipa,temp);
if(i<3)
strcat(ipadd[j].ipa,".");
}
printf("%s",itoa(j+1,x1,10));
temp1=strcat("SYSTEM",x1);
strcpy(ipadd[j].pha,temp1);
printf("\n%s\n",ipadd[j].ipa);
printf("\n%s\n",ipadd[j].pha);
}
for(j=0;j<2;j++)
printf("\n\n%s\n",ipadd[j].ipa);
for(j=0;j<2;j++)
for(i=0;i<2;i++)
if(strcmp(ipadd[i].ipa,ip[j])==0)
{
printf("\n the physical address of the given ip address %s is",ip[j]);
printf("\n\n%s\n",ph[j]);
}
getch();
for(j=0;j<2;j++)
for(i=0;i<2;i++)
if(strcmp(ipadd[i].pha,ph[j])==0)
{
printf("\n the ip address of the given physical address %s is",ph[j]);
printf("\n\n%s\n",ip[j]);
}
getch();
}
```

OUTPUT:

```
the physical address of the given ip address  135.237.105.128 is SYSTEM1

the physical address of the given ip address  225.22.205.221 is SYSTEM2

the ip address of the given physical address SYSTEM1 is 135.237.105.128

the ip address of the given physical address SYSTEM2 is 225.22.205.221
```

2. BIT STUFFING (SENDER)

AIM:

To write a program to implement bit stuffing for a given binary file. That is stuff an extra ‘0’ bit after continuous five 1’s in the file.

ALGORITHM:

Step 1: Start the program.
Step 2: Open the given input binary file in read mode and open the output file in write mode.
Step 3: Start reading the input binary file character by character and write it in the output binary file.
Step 4: Use previous, next and count to store the values of previous character, next character and number of ones respectively.
Step 5: If the count value is 5 then write one ‘0’ in the file in the current position of the output file and reset the value of count to zero.
Step 6: Continue steps 3 to 5 till the end of the file.
Step 7: Close the input and output files.
Step 8: Open the output binary file in read mode and display the characters from it.
Step 9: An extra ‘0’ bit is inserted after continuous five 1’s in the file.
Step 10: Close all the files.
Step 11: Stop the program.

<u>CODING:</u>

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fpt1, *fpt2;
char c, prev,next;
int i = 1;
clrscr();
fpt1 = fopen("in.txt","r");
fseek(fpt1, 0, SEEK_SET);
fpt2 = fopen("out.txt","w+");
        if ((c = fgetc(fpt1))!=EOF)
        {
                fputc(c,fpt2);
                prev = c;
        }
do
{
        next = fgetc(fpt1);
                if (prev == '1'&& next == '1')
                {
                        fputc(next,fpt2);
                        i++;
                }
                else
                {
                        fputc(next,fpt2);
                        i = 1;
                }
                if (i==5)
                        fputc('0',fpt2);
                else
                        {       prev = next; }
}while(next!=EOF);
fclose(fpt1);
fclose(fpt2);
fpt1 = fopen("in.txt","r");
printf("\n\nThe input file is: \n\n ");
do
{
c = fgetc(fpt1);
putch(c);
}while(c!= EOF);
fclose(fpt1);
printf("\n\nThe output file is: \n\n ");
fpt2 = fopen("out.txt","r");
do
{
c= fgetc(fpt2);
putch(c);
}while(c!=EOF);
fclose(fpt2);
getch();
return 0;
}
```

OUTPUT:

```
The input file is:
100111110111111010101011111111100110

The output file is:
1001111100111110101010101111101111100110
```

3. BIT STUFFING (RECIEVER)

AIM:

To write a program to implement bit stuffing for a given binary file. That is to remove the extra ‘0’ bit after continuous five 1’s in the file.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Open the given input binary file in read mode and open the output file in write mode.
Step 3: Start reading the input binary file character by character and write it in the output binary file.
Step 4: Use previous, next and count to store the values of previous character, next character and number of ones respectively.
Step 5: If the count value is 5 then delete the next ‘0’ character from the file in the current position of the output file and reset the value of count to zero.
Step 6: Continue steps 3 to 5 till the end of the file.
Step 7: Close the input and output files.
Step 8: Open the output binary file in read mode and display the characters from it.
Step 9: The extra ‘0’ bit is removed from the file if there is continuous five 1’s in the file.
Step 10: Close all the files.
Step 11: Stop the program.
```

CODING:

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fpt1, *fpt2;
char c, prev,next;
int i = 1;
clrscr();
fpt1 = fopen("out.txt","r");
fseek(fpt1, 0, SEEK_SET);
fpt2 = fopen("org.txt","w+");
        if ((c = fgetc(fpt1))!=EOF)
        {
                fputc(c,fpt2);
                prev = c;
        }
do
{
        next = fgetc(fpt1);
                if (prev == '1'&& next == '1')
                {
                        fputc(next,fpt2);
                        i++;
                }
                else
                {
                        fputc(next,fpt2);
                        i = 1;
                }
                if (i==5)
                {
                        c= fgetc(fpt1);
                }
                else
                {
                        prev = next;
                }
}while(next!=EOF);
fclose(fpt1);
fclose(fpt2);
fpt1 = fopen("out.txt","r");
printf("\n\nThe input file is: \n\n ");
do
{
c = fgetc(fpt1);
putch(c);
}while(c!= EOF);
fclose(fpt1);
printf("\n\nThe original input file is: \n\n ");
fpt2 = fopen("org.txt","r");
do
{
c= fgetc(fpt2);
putch(c);
}while(c!=EOF);
fclose(fpt2);
getch();
return 0;
}
```

OUTPUT:

```
The input file is:
10011111001111101010101011111011110011O
```

```
The original input file is:
100111110111111010101011111111100110
```

4. CYCLIC REDUNDANY CHECK

AIM:

To find the Cyclic redundancy check for the given data bits.

ALGORITHM:

Step 1: Start the program.
Step 2: Accept the number of bits in the input data stream (n).
Step 3: Accept the number of bits in the divisor (d).
Step 4: Accept the input bits and the divisor bits.
Step 5: Append (d-3) ‘0’ bits to the end of the input data bits.
Step 6: Perform binary division.
Step 7: Remove the appended ‘0’ bits and append the remainder bits to the original data bits.
Step 8: Again perform binary division.
Step 9: If the remainder is zero then there is no error in the transmitted bits else the received bits are in error.
Step 10: Stop the program.

<u>CODING:</u>

```
#include<stdio.h>
#include<conio.h>
void main()
{
int
arr1[20],arr2[20],divarr[10],n,d,i,j,k,l,x,z=0,temp1[]={1,1,1,1,1,1,1},temp0[]={0,0,0,0,0,
0,0},
restarr[10],restarr1[10];
printf("Enter the number bit(s) in data\n");
scanf("%d",&n);
printf("Enter the number of bit(s) in the divisor\n");
scanf("%d",&d);
printf("Enter the data\n");
for(i=0;i<n;i++)
scanf("%d",&arr1[i]);
printf("Enter the divisor\n");
for(j=0;j<d;j++)
scanf("%d",&divarr[j]);
for(i=0;i<n;i++)
{
arr2[i]=arr1[i];
}
for(i=0;i<d-1;i++)
{
arr1[n]=0;
n++;
}
```

```
for(i=0;i<n;i++)
{

        if(arr1[i]==divarr[0])
        {

            k=i;
            for(j=0;j<d;j++)
            {

                if(arr1[k]==divarr[j])
                {
                    arr1[k]=0;
                    k++;




                }
                else
                {
                    arr1[k]=1;
                    k++;
                }

            }

        }
        else if(arr1[i]==1)
        {
          for(j=0;j<d;j++)
            {
                k=i;
                if(arr1[k]==temp1[j])
                {
                    arr1[k]=0;
                    k++;
                }
                else
                {
                    arr1[k]=1;
                    k++;
                }

            }
          }
          else if(arr1[i]==0)
          {
            for(j=0;j<d;j++)
            {
                k=i;
```

```
                                        if(arr1[k]==temp0[j])
                                        {
                                                arr1[k]=0;
                                                k++;
                                        }
                                        else
                                        {
                                                arr1[k]=1;
                                                k++;
                                        }
                                }
                        }
                }
//      printf("the n value is %d",n);
x=n;
        for(i=0;i<d-1;i++)
        {
  restarr[i]=arr1[x-2];
        x++;
  }
//  printf("\n\n\nn value %d ",n);

  printf("The first remainder is\n");
  for(i=0;i<d-1;i++)
  {
  printf("%d\n",restarr[i]);
  }
  getch();

        printf("The original value is\n");
        for(i=0;i<6;i++)
        {
        printf("%d\n",arr2[i]);
        }
        x=n;
        for(i=0;i<d-1;i++)
        {
        arr2[x-3]=restarr[i];
        x++;
        }
        printf("The remainder bit is appended with the original data\n");
        for(i=0;i<n;i++)
        {
        printf("%d\n",arr2[i]);
        }
        for(i=0;i<n;i++)
      {

                if(arr2[i]==divarr[0])
                {

                        k=i;
                                for(j=0;j<d;j++)
                                {

                                        if(arr2[k]==divarr[j])
                                        {
                                                arr2[k]=0;
                                                k++;
                                        }
                                        else




                                        {
                                                arr2[k]=1;
                                                k++;
                                        }

                                }

                }
                else if(arr2[i]==1)
                {
                        for(j=0;j<d;j++)
                                {
                                        k=i;
                                        if(arr2[k]==temp1[j])
                                        {
                                                arr2[k]=0;
                                                k++;
                                        }
                                        else
                                        {
                                                arr2[k]=1;
                                                k++;
                                        }

                                }
                        }
                        else if(arr2[i]==0)
                        {
                                for(j=0;j<d;j++)
                                {
                                        k=i;
                                        if(arr2[k]==temp0[j])
                                        {
                                                arr1[k]=0;
                                                k++;
                                        }
                                        else
                                        {
                                                arr2[k]=1;
```

```
                                                k++;
                                          }
                                    }
                              }

            }

                        for(i=0;i<d-1;i++)
            {
        restarr1[i]= arr2[n-3];
        n++;
        }
        printf("The final remainder is\n");
        for(i=0;i<d-1;i++)
        {
        printf("%d\n",restarr1[i]);
        }
        for(i=0;i<d-1;i++)
        {
        if(restarr1[i]==0)
        z++;
        }
       if(z==d-1)
        printf("There is no error in the bit\n");
        getch();
}
```
OUTPUT:

```
Enter the no of bits in the dividend: 5

Enter the no of bits in the divisor: 2


Enter the dividend:
1
1
0
1
0
Enter the divisor:
1
0
The first remainder is : 1

The original value is:
1
1
0
1
0
The remainder bits appended with the original data;
```

5. SLIDING WINDOW PROTOCOL (GO BACK N)

AIM:

To write a program to perform simulation on sliding window protocol.

ALGORITHM:

Step 1: Start the program.
Step 2: Generate a random that gives the total number of frames to be transmitted.
Step 3: Set the size of the window.
Step 4: Generate a random number less than or equal to the size of the current window and identify the number of frames to be transmitted at a given time.
Step 5: Transmit the frames and receive the acknowledgement for the frames sent.
Step 6: Find the remaining frames to be sent.
Step 7: Find the current window size.
Step 8: If an acknowledgement is not received for a particular frame retransmit the frames from that frame again.
Step 9: Repeat the steps 4 to 8 till the number of remaining frames to be send becomes zero.
Step 10: Stop the program.

CODING:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int temp1,temp2,temp3,temp4,i,winsize=8,noframes,moreframes;
char c;
int reciever(int);
int simulate(int);
clrscr();
temp4=0,temp1=0,temp2=0,temp3=0;
for(i=0;i<200;i++)
rand();
noframes=rand()/200;
printf("\n number of frames is %d",noframes);
getch();
moreframes=noframes;
while(moreframes>=0)
{
temp1=simulate(winsize);
winsize-=temp1;
temp4+=temp1;
if(temp4 >noframes)
temp4 = noframes;
for(i=temp3+1;i<=temp4;i++)
printf("\nsending frame %d",i);
getch();
temp2=reciever(temp1);
temp3+=temp2;
if(temp3 > noframes)
temp3 = noframes;
printf("\n acknowledgement for the frames up to %d",temp3);
getch();
moreframes-=temp2;
temp4=temp3;
if(winsize<=0)
winsize=8;
}
printf("\n end of sliding window protocol");
getch();
}
int reciever(int temp1)
{
int i;
for(i=1;i<100;i++)
rand();
i=rand()%temp1;
return i;
}
int simulate(int winsize)
{
int temp1,i;
for(i=1;i<50;i++)
temp1=rand();
if(temp1==0)
temp1=simulate(winsize);
i = temp1%winsize;
if(i==0)
return winsize;
else
return temp1%winsize;
}
```

<u>OUTPUT:</u>

```
Number of frames: 55
Sending frame 1
Sending frame 2
Sending frame 3
            Acknowledgement for the frames upto 0
Sending frame 1
            Acknowledgement for the frames upto 0
Sending frame 1
Sending frame 3Sending frame 4
            Acknowledgement for the frames upto 4
            Acknowledgement for the frames upto 54
Sending frame 55
            Acknowledgement for the frames upto 55
End of sliding window protocol
```

6. SLIDING WINDOW PROTOCOL (SELECTIVE REPEAT)

AIM:

To write a program to perform simulation on sliding window protocol.

ALGORITHM:

Step 1: Start the program.
Step 2: Generate a random that gives the total number of frames to be transmitted.
Step 3: Set the size of the window.
Step 4: Generate a random number less than or equal to the size of the current window and identify the number of frames to be transmitted at a given time.
Step 5: Transmit the frames and receive the acknowledgement for the frames sent.
Step 6: Find the remaining frames to be sent.
Step 7: Find the current window size.
Step 8: If an acknowledgement is not received for a particular frame retransmit that frame alone again.
Step 9: Repeat the steps 4 to 8 till the number of remaining frames to be send becomes zero.
Step 10: Stop the program.

<u>CODING:</u>

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int temp1,temp2,temp3,temp4,temp5,i,winsize=8,noframes,moreframes;
char c;
int reciever(int);
int simulate(int);
int nack(int);
clrscr();
temp4=0,temp1=0,temp2=0,temp3=0,temp5 = 0;
for(i=0;i<200;i++)
rand();
noframes=rand()/200;
printf("\n number of frames is %d",noframes);
getch();
moreframes=noframes;
while(moreframes>=0)
{
temp1=simulate(winsize);
winsize-=temp1;
temp4+=temp1;
if(temp4 >noframes)
temp4 = noframes;
for(i=noframes - moreframes;i<=temp4;i++)
printf("\nsending frame %d",i);
getch();
temp2=reciever(temp1);
temp3+=temp2;
if(temp3 > noframes)
temp3 = noframes;
temp2 = nack(temp1);
temp5+=temp2;
if (temp5 !=0)
{
printf("\n No acknowledgement for the frame %d",temp5);
getch();
for(i=1;i<temp5;i++)
;
printf("\n Retransmitting frame %d",temp5);
getch();
}
moreframes-=temp1;
if(winsize<=0)
winsize=8;
}
printf("\n end of sliding window protocol Selective Reject");
getch();
}

int reciever(int temp1)
{
int i;
for(i=1;i<100;i++)
rand();
i=rand()%temp1;
return i;
}

int nack(int temp1)
{
int i;
for(i=1;i<100;i++)
rand();
i=rand()%temp1;
return i;
}
int simulate(int winsize)
{
int temp1,i;
for(i=1;i<50;i++)
temp1=rand();
if(temp1==0)
temp1=simulate(winsize);
i = temp1%winsize;
if(i==0)
return winsize;
else
```

```
return temp1%winsize;
}
```

OUTPUT:

```
Number of frames: 55

Sending frame 1
Sending frame 2
Sending frame 3
Sending frame 4
            No Acknowledgement for the frame 2
Retransmitting frame 2
Sending frame 5
Sending frame 6
            No Acknowledgement for the frame 2
Retransmitting frame 2
Sending frame 3
Sending frame 4
            No Acknowledgement for the frame 4
.
.
.
Sending frame 54
Sending frame 55
End of sliding window protocol
```

7. STOP AND WAIT PROTOCOL

AIM:

To write a program to simulate stop and wait protocol

ALGORITHM:

Step 1: Start the program.
Step 2: Generate a random that gives the total number of frames to be transmitted.
Step 3: Transmit the first frame.
Step 4: Receive the acknowledgement for the first frame.
Step 5: Transmit the next frame
Step 6: Find the remaining frames to be sent.
Step 7: If an acknowledgement is not received for a particular frame retransmit that frame alone again.
Step 8: Repeat the steps 5 to 7 till the number of remaining frames to be send becomes zero.
Step 9: Stop the program.

CODING:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int i,j,noframes,x,x1=10,x2;
clrscr();
for(i=0;i<200;i++)
rand();
noframes=rand()/200;
i=1;
j=1;
noframes = noframes / 8;
printf("\n number of frames is %d",noframes);
getch();
while(noframes>0)
{
printf("\nsending frame %d",i);
srand(x1++);
x = rand()%10;
if(x%2 == 0)
{
for (x2=1; x2<2; x2++)
{
  printf("waiting for %d seconds\n", x2);
  sleep(x2);
}
printf("\nsending frame %d",i);
srand(x1++);
x = rand()%10;
}
printf("\nack for frame %d",j);
noframes-=1;
i++;
j++;
}
printf("\n end of stop and wait protocol");
getch();
}
```

OUTPUT:

```
No of frames is 6
Sending frame 1
Acknowledgement for frame 1
Sending frame 2
Acknowledgement for frame 2
Sending frame 3
Acknowledgement for frame 3
Sending frame 4
Acknowledgement for frame 4
Sending frame 5
Waiting for 1 second
Retransmitting frame 5
Acknowledgement for frame 5
Sending frame 6
Waiting for 1 second
Sending frame 6
Acknowledgement for frame 6
End of stop and wait protocol
```

8. RS232 INTERFACE

AIM:

To write a program to transfer a files using RS232 interface.

ALGORIHTM:

Step 1: Start the program.
Step 2: Select the file to transfer.
Step 3: Identify the Port number of the sender the receiving PC's for communication.
Step 4: Find the two systems are ready to communicate.
Step 5: Read character by character and transfer the data through the RS 232 interface.
Step 6: The contents of the file will be display on the receiving system.
Step 7: Repeat the steps 5 and 6 till the end of the file.
Step 8: Stop the program

CODING:

```
/*  PC to PC Communication via RS232 port.*/

#include<conio.h>    //for getch, putch,textcolor...
#include<bios.h>     //for bioscom
#include<stdio.h>    //for file functions
#include<dos.h>

#define COM1       0
#define DATA_READY 0x100
#define SETTINGS ( 0x80 | 0x02 | 0x00 | 0x00)

void chatwindow();
int chatting();
void dsply(char *, int);
void welcome();

void main()
{
        textbackground(0);
        welcome();
        chatwindow();
        chatting();
}

void chatwindow()
{
  int i,j;
  textbackground(0);
  clrscr();
  cprintf("Esc:  Exit      Enter: Send         :Recieved msg       :Sent msg");
  gotoxy(63,1);  textcolor(14);  putch(219);
  gotoxy(43,1);  textcolor(10);  putch(219);
  window(10,3,70,20);
```

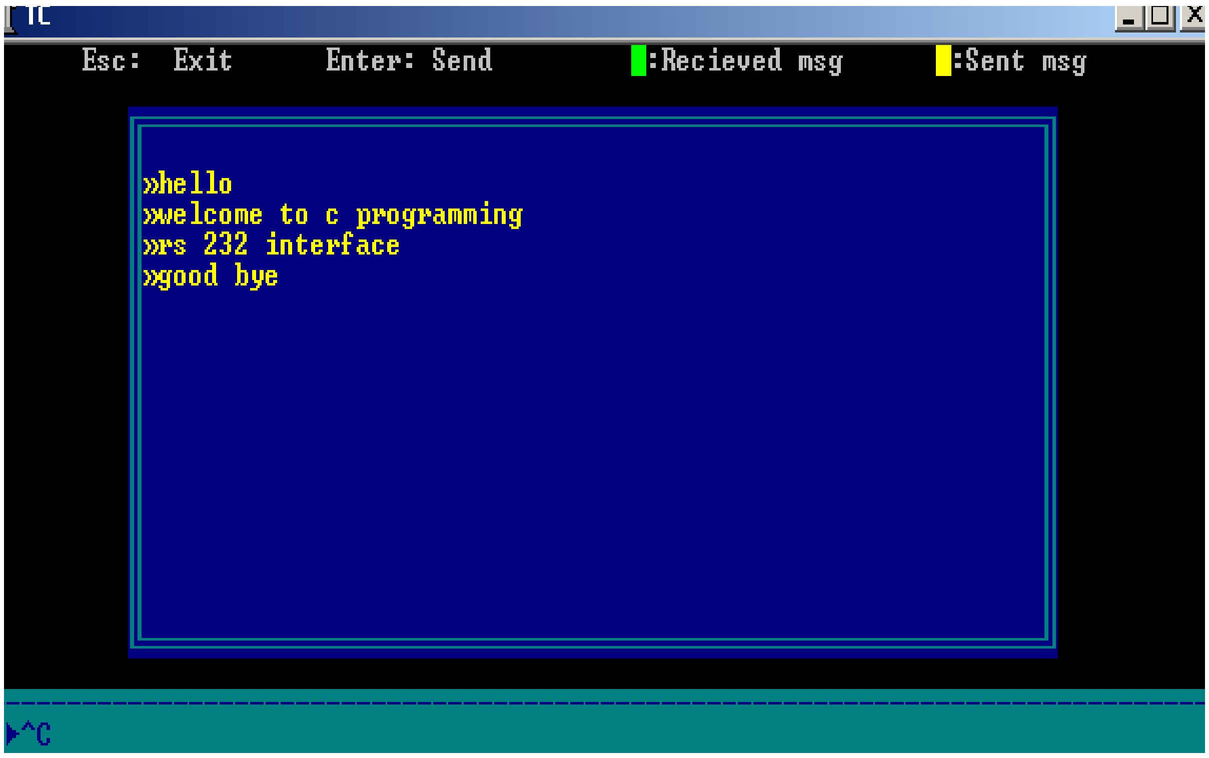
```
 textbackground(1);
 clrscr();
 textcolor(3);
 window(1,1,80,25);

 for(i=0;i<=60;i++)
 {
         gotoxy(10+i,3);     putch(205);  //Í
         gotoxy(10+i,20);    putch(205);  //Í
 }
 for(j=0;j<=16;j++)
 {
         gotoxy(10,3+j);     putch(186);  //º
         gotoxy(70,3+j);     putch(186);  //º
 }

 gotoxy(10,3); putch(201);          //É
 gotoxy(70,3);putch(187);           //»
 gotoxy(10,20); putch(200);          //È
 gotoxy(70,20);putch(188);          //¼
 textbackground(3);
 textcolor(1);                   //BLUE
 window(1,22,80,24);
 clrscr();                      //make background blue
 window(1,1,80,25);
 for(i=1;i<=80;i++)
 {
              gotoxy(i,22);      putch('-');  //196
              gotoxy(i,24);      putch('-');  //Ä
 }
 gotoxy(2,23);  putch(16);          //
}
int chatting()
{
        char msg[80],msg_in[80];
        int status,out,in,count=0,count_in=0,i,extra;
        bioscom(0, SETTINGS, COM1); //init. port
        //alternative to outportb(),perticularly for rs 232 communication.
        window(3,23,80,23);        //message entry window
        while (1)
        {
              status = bioscom(3, 0, COM1);
              if (status & DATA_READY)
              {
        if ((out = bioscom(2, 0, COM1) & 0x7F) != 0)// input message byte.
               msg_in[count_in++]=out;
        if(out==13)
        {
               dsply(msg_in,10);
               count_in=0;
               gotoxy(count+1,1);
        }
```

```
            }
            if (kbhit())   //character entered
            {
        gotoxy(count+1,1);
        if ((in = getche()) == 27)
            break;
        if(in==0)
        {
            in= getch();  //special function key pressed, ignore
            continue;
        }
        if(in==8 && count>0) { count-=2;cprintf(" \b");}     //BackSpace
        else msg[count]=in;
        if(count==70) {in=13; msg[70]=13;}
        count++;
        if(in==13 ) //Enter: Send
        {
         i=0;
         dsply(msg,14);
         for(i=0;i<count;i++)
         {
              bioscom(1, msg[i], COM1);
         }
         clrscr();
         count=0;
        }
         }
        }
         return 1;
}
void dsply(char str[80],int col)
{
        int i=0;
        static int line=1;
        window(11,4,69,19);       //Chat display window
        textbackground(1);        //BLUE
        textcolor(col);
        gotoxy(1,line);
         putch('\n');
         putch(175);
         while(str[i]!=13)
         {
        putch(str[i]);
        i++;
         }
         line=wherey();       //store present line
         window(3,23,80,23);  //message enter window
         textbackground(3);
         textcolor(1);
}
```

```
void welcome()
{
  clrscr();
  gotoxy(3,12);
  cprintf("Press any key to start....");
  window(1,1,80,25);
  getch();
}
```

OUTPUT:

```
  Press any key to start....
```



```
Esc:  Exit        Enter: Send         🟩:Recieved msg        🟨:Sent msg

»hello
»welcome to c programming
»rs 232 interface
»good bye

^C
```

9. CLIENT SERVER CHAT APPLICATION

AIM:

To write a program to perform client server chat application.

ALGORITHM:

Step 1: Start the program.
Step 2: Establish the connection between the client and the server.
Step 3: Identify the Server address, family and port number.
Step 4: Create Socket to send the message
Step 5: Bind the connection
Step 6: Confirm the connection between the client and the server
Step 7: Create a buffer to send and receive the message
Step 8: Start sending the messages to and from the client and server
Step 9: Repeat the steps 4 to 8 until the connection is terminated.
Step 10: Stop the program.

<u>CODING:</u>

<u>CLIENT PROGRAM:</u>

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5003
int main(int argc, char *argv[])
{
int sockfd;
struct sockaddr_in serv_addr;
struct hostnet *server;
char buffer[4096];
sockfd = socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(SERV_TCP_PORT);
connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
printf("\nenter the message:\n");
bzero(buffer,4096);
printf("\nclient:");
fgets(buffer,4096,stdin);
write(sockfd,buffer,4096);
bzero(buffer,4096);
read(sockfd,buffer,4096);
printf("\nserver msg :%s",buffer);
printf("\n");
close(sockfd);
return 0;
}
```

SERVER PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5003
int main(int argc, char *argv[])
{
int sockfd, newsockfd, clength;
struct sockaddr_in serv_addr, cli_addr;
char buffer[4096];
sockfd = socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(SERV_TCP_PORT);
bind(sockfd,(struct sockaddr *)&serv_addr, sizeof(serv_addr));
listen(sockfd,5);
clength = sizeof(cli_addr);
newsockfd = accept(sockfd,(struct sockaddr *)&cli_addr, &clength);
bzero(buffer,4096);
read(newsockfd,buffer,4096);
printf("\nclient message: %s",buffer);
write(newsockfd,buffer,4096);
printf("\n");
close(sockfd);
return 0;
}
```

OUTPUT:

CLIENT SIDE:

```
Enter the message:

Client:

Hai

Server message:

Hai
```

SERVER SIDE:

```
Client message:

Hai
```

10.FIND THE PORT ADDRESS FOR SERVICE NAME

AIM

To find the port number of service by giving service name

ALGORITHM

```
Step1: Start
Step2: Create a pointer for the structure of  servent  (ser)
Step3: Call getservbyname  function and pass a service name as a parameter.
          It  returns the pointer to the structure of servent which is stored in the 'ser'
Step4:Print the service name and port number of service
Step5:Terminate the program
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet.h/in.h>
int main(int argc,char *argv[1])
{
struct servent *ser;    // pointer
if (argc!=2)
   {
   fprintf( stderr, "Enter the Service Name");
   exit(1);
   }
ser=getservbyname(argv[1],NULL);  // **function//
if(ser==NULL)
  {
     fprintf(stderr,"Service not found\n");
  }
printf("Service name is %s\n",ser->s_name); // print the service name
printf("Port number is %d\n",ntohs(ser->s_port)); // print the port number
}
```

INPUT

```
Service name is http
Portnumber  is 80
```

RESULT

Thus the program for finding the port number by giving the service name is displayed and verfied.

11.OSPF PROTOCOL

AIM:

ALGORITHM:

Step1: Input the number of nodes
Step2: Get the distance between two nodes
Step3: Get the value in matrix form
Step4: Compare the distance between the nodes
Step: If the addition of two distances is less than the specified distance then print the new distance.

CODING:

```
#include<stdio.h>
int main()
{
 int n,i,j,k,a[10][10],b[10][10];
 printf("enter the no of nodes:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
     if(i!=j)
     {
           printf("enter the distance b/w the host %d %d:",i+1,j+1);
           scanf("%d",&a[i][j]);

     }
 }
 for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
   {
     printf("%d\t",a[i][j]);
   }
   printf("\n");
 }
for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
   {
    b[i][j]=a[i][j];
    if(i==j)
    {
     b[i][j]=0;
    }
   }
  }
 for(k=0;k<n;k++)
 {
   for(i=0;i<n;i++)
   {
     for(j=0;j<n;j++)
     {
       if(i!=j)
            if(a[i][j]>a[i][k]+a[k][j])
                b[i][j]=a[i][k]+a[k][j];
      } printf("the o/p matrix is :\n");
 for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
   {
     printf("%d\t",b[i][j]);
   }
   printf("\n");
 }
}
   }
 }
```

OUTPUT

```
[itlab@ubuntu~]$ cc ospf.c
[itlab@ubuntu~]$ ./a.out
enter the no of nodes:3
enter the distance b/w the host 1 2:2
enter the distance b/w the host 1 3:3
enter the distance b/w the host 2 1:4
enter the distance b/w the host 2 3:10
enter the distance b/w the host 3 1:5
enter the distance b/w the host 3 2:9
0       2       3
4       0       10
5       9       0
the o/p matrix is :
0       2       3
4       0       7
5       7       0
```

12.STUDY OF NS-2

AIM:

To study about Network Simulator-2 and its uses.

Ns-2 is the most popular simulation tool for sensor networks. It began as ns (Network Simulator) in 1989 with the purpose of general network simulation. Ns-2 is an object-oriented discrete event simulator; its modular approach has effectively made it extensible. Simulations are based on a combination of C++ and OTcl. In general, C++ is used for implementing protocols and extending the ns-2 library. OTcl is used to create and control the simulation environment itself, including the selection of output data. Simulation is run at the packet level, allowing for detailed results. Ns-2 sensor simulation is a modification of their mobile ad hoc simulation tools, with a small number of add-ons. Support is included for many of the things that make sensor networks unique, including limited hardware and power. An extension developed in 2004 by [4] allows for external phenomena to trigger events. Ns-2 extensibility is perhaps what has made it so popular for sensor networks. In addition to the various extensions to the simulation model, the object-oriented design of ns-2 allows for straightforward creation and use of new protocols. The combination of easy in protocol development and popularity has ensured that a high number of different protocols are publicly available, despite not be included as part of the simulator's release. Its status as the most used sensor network simulator has also encouraged further popularity, as developers would prefer to compare their work to results from the same simulator. Ns-2 does not scale well for sensor networks. This is in part due to its object-oriented design. While this is beneficial in terms of extensibility and organization, it is a hindrance on performance in environments with large numbers of nodes. Every node is its own object and can interact with every other node in the simulation, creating a large number of dependencies to be checked at every simulation interval, leading to an n^2 relationship. Another drawback to ns-2 is the lack of customization available. Packet formats, energy models, MAC protocols, and the sensing hardware models all differ from those found in most sensors. One last drawback for ns-2 is the lack of an application model. In many network environments this is not a problem, but sensor networks often contain interactions between the application level and the network protocol level.

<u>STUDY OF TCP USING NETWORK SIMULATOR-2</u>

TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are the two most popularly used transport layer protocols in the Internet. TCP was designed to provide connection-oriented service, so data can be delivered in a reliable manner. On the other hand, UDP provides a connectionless service. Due to the unreliable features of the Internet (e.g., packet loss, channel error, broken link, etc.), UDP cannot provide a guaranteed reliable service. However, UDP, as a lightweight protocol, has the advantages of low delay and low overhead, which are attractive to some real-time applications. What will happen when TCP flows meet UDP flows in the same network? Can they fairly share the link? How do they act in case of congestion? In this assignment, we will try to answer these questions using simulation.

There are a number of commonly used network simulators, such as NS-2, GlomoSim, OpNet, and PDES. Among them, NS-2 is the most widely accepted especially by academic community. NS-2 is an ongoing open source project. The core implementation is done in C++. However, in the assignment, you are NOT asked to modify the core simulator, therefore, you do not need your C++ knowledge too much here. NS-2 uses OTCL/TK, a script language, to provide user interfaces. In order to operate on NS-2, you need to know some basic syntax of TCL/TK. If you do not have yet, you can be prepared in tutorials. Another useful resource is the webpage for NS-2: Simulations done on either Solaris or Linux will be acceptable. Once you feel reasonably confident with using NS, you can start with the following assignment:

1. Create the topology shown on next page.
2. Establish a TCP connection between n0 and n6 as shown.
3. Use the Tahoe version of TCP.
4. Set the maximum congestion window size of the TCP flow to 32 packets.
5. Establish a UDP connection between n1 and n7 as shown.
6. Each simulation should run for 200 simulation seconds.
7. Set the packet size to 512 bytes.
8. Both the UDP source and TCP source should be set to send infinite packets.
9. Set the UDP source to send packets at a rate of 200K bits per second.
10. Run the simulation and:

a. Measure the Average Bandwidth* available to the TCP flow

b. Measure the Ratio of TCP/UDP bytes received.

* Average Bandwidth of the TCP flow is defined as the ratio of the total bytes of TCP packets received to the total simulation time.

11. Repeat the process for rates of UDP flow from 200K bits per second up to 4M bits per second in increments of 380K bits.

12. Plot graphs for

a. Average Bandwidth of TCP flow vs. Packet rate of UDP flow.

b. Ratio of TCP/UDP bytes received vs. Packet rate of UDP flow.

13. Repeat the experiments with TCP Reno, TCP NewReno, and TCP Vegas. So, you need to plot 4 graphs for Average Bandwidth of TCP flow vs. Packet rate of UDP flow with different TCP versions, and 4 graphs for Ratio of TCP/UDP bytes received vs. Packet rate of UDP flow for different TCP version.

14. Change the packet rate of UDP flow to 200K bits per second and 4M bits per second, repeat with TCP Tahoe, Reno, New Reno, and Vegas, and plot the Congestion Window Size of the TCP flow v/s Time.

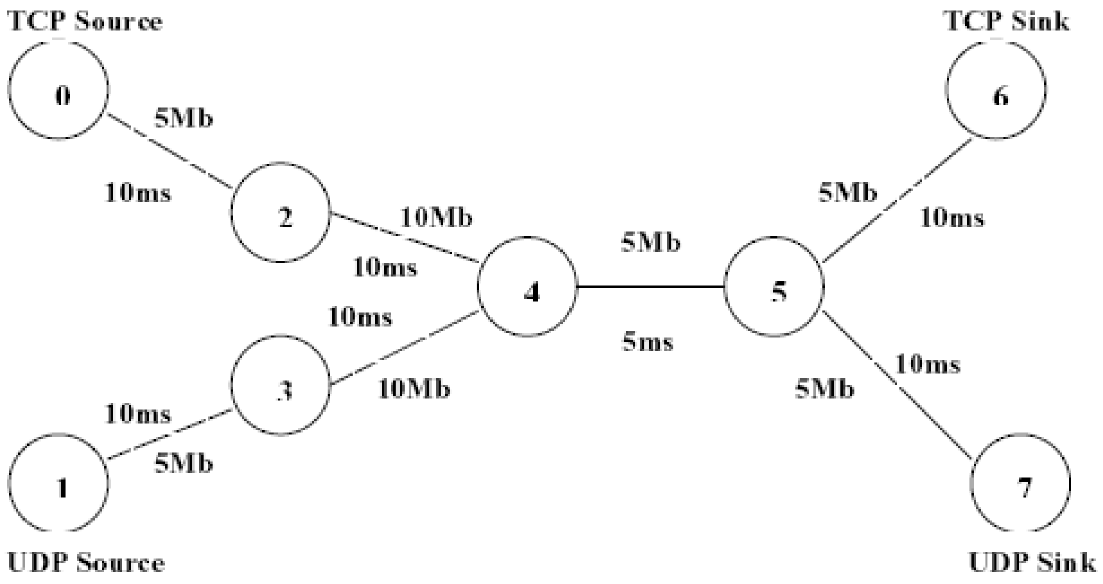
15. Observations to be made:

a. Comment on the difference in the average bandwidths available to different versions of TCP. Explain the difference in terms of the congestion control mechanisms of these versions. Which version would you like to adopt for your network?

b. Comment on the bandwidth share of TCP and UDP. Do you think that UDP traffic is suppressing the bandwidth share of TCP traffic? If yes, can you suggest a way to prevent that?

c. Based on graphs from question 14, discuss the differences of TCP Tahoe, Reno, New Reno, and Vegas.

d. Comment on the reliability of TCP and UDP traffic. Under what circumstances would you use UDP?


TCP Source
0
5Mb
10ms
2
10Mb
10ms
4
5Mb
5ms
5
10ms
3
10Mb
10ms
5Mb
1
UDP Source
TCP Sink
6
5Mb
10ms
10ms
5Mb
7
UDP Sink

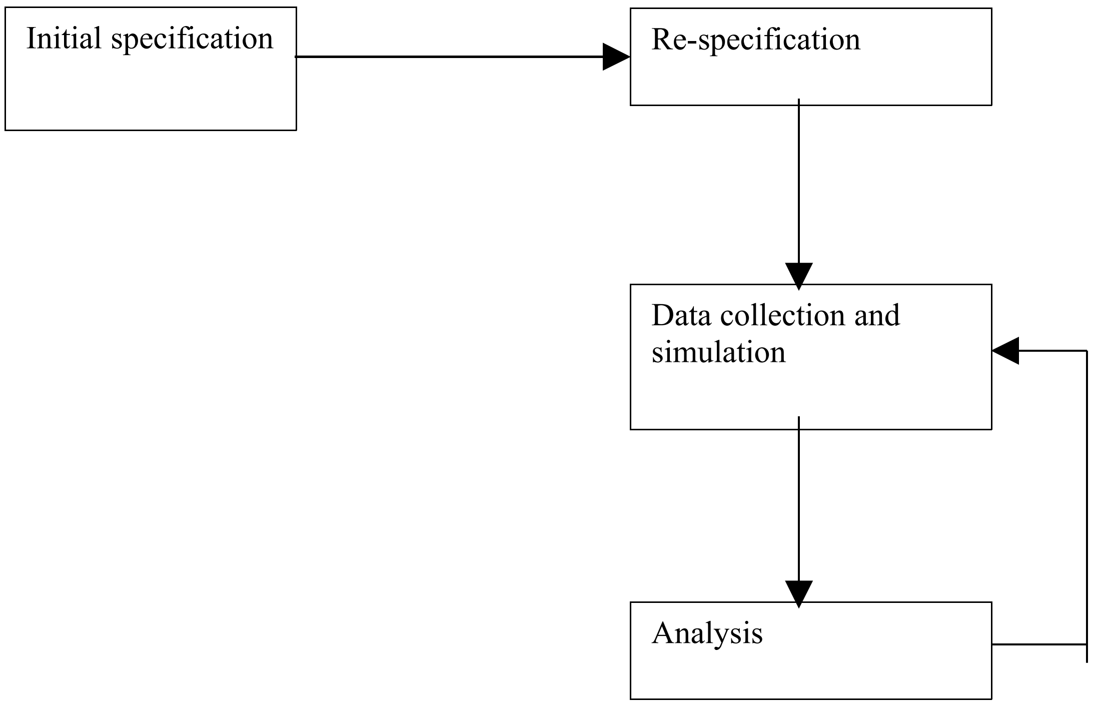
13.STUDY OF GLOMOSIM/OPNET ARCHITECTURES

AIM:

To study about Glomosim/OPNET architectures and their uses

About OPNET Architecture

OPNET provides a comprehensive development environment for modeling and performance-evaluation of communication networks and distributed systems. The package consists of a number of tools, each one focusing on particular aspects of the modeling task. These tools fall into three major categories that correspond to the three phases of modeling and simulation projects: *Specificatio*n, *Data Collection and Simulatio*n, and *Analysi*s. These phases are necessarily performed in sequence. They generally form a cycle, with a return to Specification following Analysis. Specification is actually divided into two parts: initial specification and re-specification, with only the latter belonging to the cycle, as illustrated in the following figure.



MODEL SPECIFICATION

Model specification is the task of developing a representation of the system that is to be studied. OPNET supports the concept of model reuse so that most models are based on lower level models developed beforehand and stored in model libraries. Ultimately, however all models are based on the basic concepts and primitive building blocks supplied by the OPNET environment.

Specification Editors

OPNET supports model specification with a number of tools or *editors* that Capture the characteristics of a modeled system's behavior. Because it is based on a suite of editors that address different aspects of a model, OPNET is able to offer specific capabilities to address the diverse issues encountered in networks and distributed systems. To present the model developer with an intuitive interface, these editors break down the required modeling information in a manner that parallels the structure of actual network systems. Thus, the model-specification editors are organized in an essentially hierarchical fashion. Model specifications performed in the Project Editor rely on elements specified in the Node Editor; in turn, when working in the Node Editor, the developer makes use of models define ding to the cycle.

OPNET is another discrete event, object-oriented, general-purpose network simulator. It uses a hierarchical model to define each aspect of the system. The top level consists of the network model, where topology is designed. The next level is the node level, where data flow models are defined. A third level is the process editor, which handles control flow models. Finally, a parameter editor is included to support the three higher levels. The results of the hierarchical models are an event queue for the discrete event simulation engine and a set of entities representing the nodes that will be handling the events. Each entity in the system consists of a finite state machine, which processes the events during simulation. OPNET is capable of recording a large set of user defined results Unlike ns-2 and GloMoSim, OPNET supports the use of modeling different sensor-specific hardware, such as physical-link transceivers and antennas. OPNET can also be used to define custom packet formats. The simulator aids users in developing the various models through a graphical interface. The interface can also be used to model, graph, and animate the resulting output. OPNET suffers from the same object-oriented Scalability problems as ns-2. It is not as popular as ns-2 or GloMoSim, at least in research being made publicly available, and thus does not have the high number of protocols available to those simulators. Additionally, OPNET is only available in commercial form.

OPNET can also be used to define custom packet formats. The simulator aids users in developing the various models through a graphical interface. The interface can also be used to model, graph, and animate the resulting output. OPNET suffers from the same object-oriented Scalability problems as ns-2. It is not as popular as ns-2 or GloMoSim, at least in research being made publicly available, and thus does not have the high number of protocols available to those simulators. Additionally, OPNET is only available in commercial form.

Study of GloMoSim

GloMoSim was developed in 1998 for mobile wireless networks. It is written in Parsec, which is an extension of C for parallel programming. The ability to use GloMoSim in a parallel environment distinguishes it from most other sensor network simulators. Because of communication that must take place between simulated nodes on different machines, parallel network simulation is difficult. The use of a wireless model further complicates things; communication between simulated nodes is more complicated and computationally intensive than in a wired simulated network. Additionally, the use of broadcast communication protocols, common in mobile networks, leads to significantly more traffic between computers on the distributed network. Like ns-2, GloMoSim is designed to be extensible, with all protocols implemented as modules in the GloMoSim library. The GloMoSim architecture is composed of a number of different layers. Each layer uses a different set of protocols and has an API defined to communicate with the surrounding layers. Like ns-2, GloMoSim uses an object-oriented approach. However, the designers realized that a purely object-oriented approach would not be scalable. Instead, GloMoSim partitions the nodes, and each object is responsible for running one layer in the protocol stack of every node for its given partition. This helps to ease the overhead of a large network. GloMoSim still contains a number of problems.

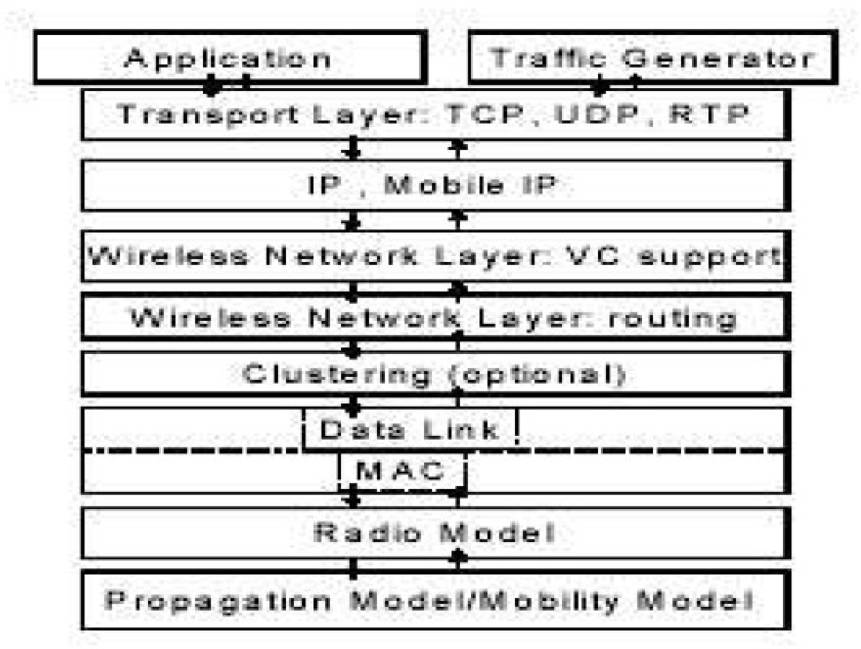

Application
Traffic Generator
Transport Layer: TCP, UDP, RTP
IP , Mobile IP
Wireless Network Layer: VC support
Wireless Network Layer: routing
Clustering (optional)
Data Link
MAC
Radio Model
Propagation Model/Mobility Model

Figure 1: GloMoSim Architecture

GloMoSim still contains a number of problems. While effective for simulating IP networks, it is not capable of simulating any other type of network. This effectively ensures that many sensor networks cannot be simulated accurately. Additionally, GloMoSim does not support phenomena occurring outside of the simulation environment, all events must be generated from another node in the network. Finally, GloMoSim stopped releasing updates in 2000. Instead, it is now updated as a commercial product called QualNet

.