C++ Simulator for IC-IoT under Buffer-Space Sharing Policy

Macro Define

```
#ifndef MACRO_DEF_H_INCLUDED
#define MACRO_DEF_H_INCLUDED

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define PI 3.141592654



#endif // MACRO_DEF_H_INCLUDED

```

Structure Define

```cpp
#ifndef MY_STRUC_H_INCLUDED
#define MY_STRUC_H_INCLUDED

#include <queue>

using namespace std;

/**************************data structure************************/

struct Packet
{
     int  id;              //packet indicator
     int  arrival_time ;   //arrival time of this packet
     int  queue_head_time;
     int  reception_time ; // reception time of this packet
     //the time when this packet arrives its local queue
};

struct Node
{
     int  row ; //the row id of this node in a m*m cell-partition network, the cell number C=m*m
     int  col ; //the column id of this node

     queue <struct Packet >local_queue ;   //store locally generated packets
     queue <struct Packet > *relay_queue ; //n-2 relay queues to store packets for other traffic flows
     int  source_queue_length      ; //
     int  sum_relay_queue_length       ;


     int  arrival_ct ; //total self-generated packets at this node
     int  recv_ct ;     //total received packets at this node as a destination
};



struct Cell
{
     int  row; //the row id of this cell
     int  col; //the column id of this cell

     int  *node_in_cell; //recording the node id in this cell;
     int  nodenum_of_cell; //recording the node number of this cell;
};

#endif // MY_STRUC_H_INCLUDED

```

Function Declaration

```
#ifndef MY_FUNC_H_INCLUDED
#define MY_FUNC_H_INCLUDED

float ran0_1(long  *idum);

int probabilityP(float p);

void update_node_position_IID(int n,int m, struct Node *node);

void update_node_position_RWalk(int n,int m, struct Node *node);

void update_node_position_RWaypoint(int n,int m, struct Node *node);

void collect_nodes_per_cell(int n, int m, struct Node *node);

void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id);

void SRtrans(struct Node *node, long time_slot,int trans_id, int recv_id);

void RDtrans(struct Node *node, long time_slot,int trans_id, int recv_id);

void THROR(struct Node *node, long time_slot);

#endif // MY_FUNC_H_INCLUDED

```

Main Function

```
/*************Buffer Space Sharing*********************/

/****************System Header Files*********************/
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <string>

/*****************My Header Files***********************/
#include "macro_def.h"
#include "my_func.h"
#include "my_struc.h"


/*****************global variables*********************/

long   seed [1];

struct Cell **cell;

int buffer_bound;

float alpha;

int n;
int m;

int K;

int tagged_S=5;
int tagged_D=6;


/******************test quantities*******************/


int SD_num; //For testing the tagged flow direct transmission opportunity
int SR_num; //For testing the tagged flow S->R transmission opportunity
int RD_num; //For testing the tagged flow R->D transmission opportunity

int S_out_number; //For testing the output opportunity of the local queue
int R_in_number; //For testing the input rate of the relay queue
int R_out_number; //For testing the output rate of the relay queue
long   lost_number;

long   tagged_S_source_queue_empty;
long   tagged_D_source_queue_empty;

long   tagged_S_buffer_empty;
long   tagged_D_buffer_empty;

long   tagged S buffer full;
long   tagged_D_buffer_full;

float delay;
float queuing delay;
float delivery delay;
/*******************data collection*******************/

ofstream ftest;


/******************Notice! Project Entrance************/
int main()
{
    *seed =-time(0);

    /****************data collection*******************/
    ftest.open("data.dat");


    /****************network settings*****************/
    n=18;
    m=3;

    buffer bound=10;

    alpha=0.5;

    //float mus=0.153;

    cout<<"n: "<<n<<" m: "<<m<<endl;

    /*******************simulation settings***********/
    K=100;
    long   time_max[K+1];
    time_max[0]=0;
    for(int i=1; i<=K; i++)
    {
        time_max[i]=200000000;
    }


    float lambda[K+1];
```

```
97      lambda[0]=0;
98      for(int i=1; i<=K; i++)
99      {
100         lambda[i]=0.01*i;
101     }
102
103     long time_slot=-1; //slot 0, slot 1, slot 2, ......
104     int round=1; //for each network setting, how many rounds a simulation has done
105
106
107     /***************building and initializing data structure******************/
108
109     cout<< "System is allocating RAM resource for simulation!!!"<<endl<<endl;
110
111     struct Node *node; //building nodes in the network
112     node=new struct Node[n+1];//node[1],...,node[n]
113
114     cell=new struct Cell *[m]; //
115     for(int i=0; i<m; i++)
116     {
117         cell[i]= new struct Cell[m];
118         for(int j=0; j<m; j++)
119         {
120             cell[i][j].node_in_cell=new int[n+1];
121             cell[i][j].row=i;
122             cell[i][j].col=j;
123         }
124     }
125
126     /***************relay queue for other n-2 flows, the map between node[i].relay_queue[j] and its destined
127     node_id is like this: if, node_id<the current node id index i, then, the current queue index j indicates
128     the destined node_id; else if, node_id>index i, then, j=node_id-2**********************************/
129
130     for(int i=1; i<=n; i++)
131         node[i].relay_queue= new queue<struct Packet>[n-1];  //relay_queue[1], relay_queue[2],..., relay_queue[n-2]
132
133     string mobility_model;
134     mobility_model="IID";
135
136     cout<<"*************************"<<mobility_model<<"*********************************" <<endl;
137     cout<<" n="<<n<<" m="<<m<<" buffer-size="<<buffer_bound<<" transmission-ratio="<<alpha<<" No Feedback"<<endl;
138     cout<<"*****************************************************************"  <<endl<<endl;
139
140     ftest<<"*************************"<<mobility_model<<"*********************************" <<endl;
141     ftest<<" n="<<n<<" m="<<m<<" buffer-size="<<buffer_bound<<" transmission-ratio="<<alpha<<" No Feedback"<<endl;
142     ftest<<"*****************************************************************"  <<endl<<endl;
143
144     /************we do simulation as input rate lambda approaches capacity mu*******************/
145     /********************we just focus on a tagged node pair************************************/
146
147     for(round=1; round<=K; round++)
148     {
149         //simulation initialization
150
151         cout<<"************new round:"<<round<<" lambda: "<<lambda[round]<<"************"<<endl;
152         //cout<<"************new round:"<<round<<" lamda: "<<lambda<<endl;
153
154         //ftest<<"************new round:"<<round<<" lamda: "<<lambda<<endl;
155
156         ftest<<"************new round:"<<round<<" lambda: "<<lambda[round]<<"************"<<endl;
157
158
159         cout<<" System is initializing simulation status for round: "<<round<<endl;
160
161         //initialize test statistic variables
162
163
164         SD num=0;
165         SR num=0;
166         RD num=0;
167
168
169         S out number=0;
170         R in number=0;
171         R out number=0;
172         lost number=0;
173
174         queuing delay=0;
175         delivery delay=0;
176         delay=0;
177
178         tagged S source queue empty=0;
179         tagged_D_source_queue_empty=0;
180         tagged S buffer empty=0;
181         tagged_D_buffer_empty=0;
182         tagged S buffer full=0;
183         tagged_D_buffer_full=0;
184
185         //initialize each node
186         for(int i=1; i<=n; i++)
187         {
188             node[i].row=-1; //node position
189             node[i].col=-1;
190
191             //clear all queues
192             while(!(node[i].local_queue.empty())) //clear local queue
```

```
                node[i].local_queue.pop();

            for(int j=0; j<n-1; j++) //clear relay queue
            {
                while(!(node[i].relay_queue[j].empty()))
                    node[i].relay_queue[j].pop();
            }
            node[i].source_queue_length=0;
            node[i].sum_relay_queue_length=0;

            node[i].arrival_ct=0;
            node[i].recv_ct=0;
        }


        //initialize time clock
        time_slot=-1;

        cout<< " Simulation is starting !!! "<<endl;

        //the main simulation body begins here
        while(time_slot<time_max[round])
        {
            time_slot++;

            update_node_position_IID(n, m, node);
            //update_node_position_RWalk(n, m, node);
            //update_node_position_RWaypoint(n, m, node);

            collect_nodes_per_cell(n, m, node);

            THROR(node, time_slot); //包在时隙末到达

            //we locally generate packets for all n source nodes, only when time_slot==next_arrival_time
            for(int i=1; i<=n; i++)
            {

                if(probabilityP(lambda[round]))//a new packet arrives in this time slot for node i
                {
                    node[i].arrival_ct++;

                    if(node[i].source_queue_length+node[i].sum_relay_queue_length<buffer_bound)
                    {
                        struct Packet packet;
                        packet.id=node[i].arrival_ct;
                        packet.arrival_time=time_slot;
                        packet.reception_time=0;

                        if(node[i].local_queue.empty())
                        {
                            packet.queue_head_time=time_slot;
                        }
                        else
                        {
                            packet.queue_head_time=0;
                        }

                        node[i].local queue.push(packet);
                        node[i].source queue length++;
                    }
                    else
                    {
                        if(i==tagged S)
                        {
                            lost_number++;
                        }
                    }
                }
            }

            //H2HR(node, time_slot); //包在时隙初到达

            if(node[tagged S].source queue length==0)
            {
                tagged S source queue empty++;
            }

            if(node[tagged D].source queue length==0)
            {
                tagged D source queue empty++;
            }

            if(node[tagged_S].source_queue_length+node[tagged_S].sum_relay_queue_length==0)
            {
                tagged_S_buffer_empty++;
            }

            if(node[tagged_D].source_queue_length+node[tagged_D].sum_relay_queue_length==0)
            {
                tagged_D_buffer_empty++;
            }

            if(node[tagged_S].source_queue_length+node[tagged_S].sum_relay_queue_length==buffer_bound)
            {
                tagged_S_buffer_full++;
            }
```

```
289
290                 if(node[tagged_D].source_queue_length+node[tagged_D].sum_relay_queue_length==buffer_bound)
291                 {
292                     tagged_D_buffer_full++;
293                 }
294
295             }
296
297
298             cout<<" Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
299
300             cout<<" Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
301
302             cout<<" Average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
303
304
305             ftest<<" Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
306
307             ftest<<" Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
308
309             ftest<<" Average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
310
311         }
312
313         cout<< " System is deleting RAM resources!!!"<<endl<<endl;
314
315         for(int i=0; i<m; i++)
316         {
317             for(int j=0; j<m; j++)
318                 delete []cell[i][j].node_in_cell;
319         }
320
321         for(int i=0; i<m; i++)
322             delete []cell[i];
323         delete []cell;
324
325         for(int i=1; i<=n; i++)
326             delete []node[i].relay_queue;
327         delete []node;
328
329
330         ftest.close();
331
332         cout << "Simulation is finished!" <<endl;
333         int tmp=0;
334         cin>>tmp;
335
336         return 0;
337     }
338
```

Update Device Position Based on Mobility Model

```c
#include "my_func.h"
#include "my_struc.h"

extern long seed[1];


/**
update the position of each node at the beginning of each time slot
according to the i.i.d mobility model
**/

void update_node_position_IID(int n,int m, struct Node *node)
{
    int row =0, col =0;

    for(int i=1; i<=n; i++)
    {
        row=(int)(m*ran0_1(seed)); // random select a row id among 0,1,...,m-1 with equal probability
        col=(int)(m*ran0_1(seed)); // random select a column id among 0,1,...,m-1 with equal probability
        node[i].row=row;
        node[i].col=col;
    }
}


/**
update the position of each node at the beginning of each time slot
according to the random walk mobility model
**/
void update_node_position_RWalk(int n,int m, struct Node *node)
{
    int horizontal_move=0; //-1,0,1
    int vertical_move=0;//-1,0,1

    for(int i=1; i<=n; i++)
    {
        horizontal_move=(int)(3*ran0_1(seed))-1;
        vertical_move=(int)(3*ran0_1(seed))-1;
        node[i].row=(node[i].row+vertical_move+m)%m;
        node[i].col=(node[i].col+horizontal_move+m)%m;
    }
}

/**
update the position of each node at the beginning of each time slot
according to the random way point mobility model
**/
void update_node_position_RWaypoint(int n,int m, struct Node *node)
{
    int v_x=0,v_y=0; //velocity
    int d_x=0,d_y=0; //direction

    for(int i=1; i<=n; i++)
    {
        //determine move direction
        d_x=(int)(2*ran0_1(seed));//0,1
        d_y=(int)(2*ran0_1(seed));//0,1
        if(d_x==0)d_x=-1;
        if(d_y==0)d_y=-1;

        //determine speed
        v_x=(int)(3*ran0_1(seed))+1;//1,2,3
        v_y=(int)(3*ran0_1(seed))+1;//1,2,3

        node[i].row=(node[i].row+d_y*v_y+m)%m;
        node[i].col=(node[i].col+d_x*v_x+m)%m;
    }
}


```

Collect Device Information

```
#include "my_struc.h"

extern struct Cell **cell ;

/**
collect nodes in each cell
**/
void collect_nodes_per_cell(int n, int m, struct Node *node )
{
    //reset the state of each cell
    for(int i=0; i<m; i++)
    {
        for(int j=0; j<m; j++)
            cell [i][j]. nodenum_of_cell=0;
    }

    int r_d =0; //recording the difference between the node row id and the cell row id
    int c_d=0; //recording the difference between the node column id and the cell column id

    int flag=0; // 0: the node is not in this cell 1: the node is the cell

    int index=0; // recording the node is in this cell

    for(int i=1; i<=n; i++)
    {
        flag=0;

        for(int s=0; s<m; s++)
        {
            for(int t=0; t<m; t++)
            {
                     r_d =node [i]. row -cell [s][t]. row ;
                     c_d=node [i]. col-cell [s][t]. col;

                     if((r_d ==0)&&(c_d==0)) //node i is in an active cell
                     {
                    index=cell [s][t]. nodenum_of_cell;
                    cell [s][t]. node_in_cell[index]=i;
                    cell [s][t]. nodenum_of_cell++;

                    flag=1;
                    break;
                     }
            }

            if(flag==1)
                     break;
        }
    }
}

```

Two-hop Relaying Opportunistic Routing Algorithm

```
#include "my_struc.h"
#include "my_func.h"

extern long seed[1];

extern struct Cell **cell;

extern int tagged_S;
extern int tagged_D;

extern int n;
extern int m;


extern int SD_num;
extern int SR_num;
extern int RD_num;
extern int S_out_number;

extern float alpha;

/*****************************************************************************************
                                  Traffic Setting
                1-->2, 2-->3,...,i-1-->i,...,n-1-->n, n-->1
         with out loss of generality,we focus on a tagged node pair
*****************************************************************************************/

void THROR(struct Node *node, long time_slot)
{


    int dest_id=0; //indicate the direct destination node id
    int trans_id=0; //randomly selected transmitter
    int recv_id=0; //randomly selected receiver
    int index=0;

    int flag=0;

    int nodenum_of_cell=0;



    for(int i=0; i<m; i++)
    {
        for(int j=0; j<m; j++)
        {

            flag=0;

            nodenum of cell=cell[i][j].nodenum of cell;

            if(nodenum_of_cell>=2)
            {
                index=(int)(nodenum of cell*ran0 1(seed));

                trans_id=cell[i][j].node_in_cell[index];

                if(trans id==n)
                    dest id=1;
                else
                    dest_id=trans_id+1;


                for(int tmp=0; tmp<nodenum of cell; tmp++)
                {
                    if(cell[i][j].node in cell[tmp]==dest id)
                    {
                        SDtrans(node,time slot,trans id,dest id);

                        if(trans id==tagged S)
                        {
                            S out number++;
                            SD num++;
                        }

                        flag=1;
                        break;
                    }
                }

                if(flag==0)
                {
                    do
                    {
                        index=(int)(nodenum_of_cell*ran0_1(seed));
                        recv id=cell[i][j].node in cell[index];
                    }
                    while(recv_id==trans_id);

                    /***************transmission scheduling: with probability alpha, do S->R, with probability
1-alpha, do R->D*************/

                    if(probabilityP(alpha))                         //do S->R
                    {
                        SRtrans(node,time_slot,trans_id,recv_id);
                        if(trans_id==tagged_S)
```

```
                        {
                            SR_num++;
                            S_out_number++;
                        }

                    }


                    else //do R->D
                    {
                        RDtrans(node,time_slot,trans_id,recv_id);
                        if(trans_id==tagged_S)
                            RD_num++;
                    }


                }

            }
        }
    }

}

```

Source-to-Destination Operation

```
#include "my_struc.h"

extern int tagged_S;
extern float delay;
extern float queuing_delay;
extern float delivery_delay;

void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id)
{
    if(!(node[trans_id].local_queue.empty())) //if the local queue has packet
    {
        node[trans_id].local_queue.front().reception_time=time_slot; //recording the reception time of this packet

        if(trans_id==tagged_S) //without loss of generality, we focus on a tagged SD pair
        {
            struct Packet packet;
            packet=node[trans_id].local_queue.front();


queuing_delay=queuing_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.queue_head_time-packet
.arrival_time)/(node[dest_id].recv_ct+1);

delivery_delay=delivery_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packe
t.queue_head_time)/(node[dest_id].recv_ct+1);

delay=delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
ode[dest_id].recv_ct+1);

        }

        node[trans_id].local_queue.pop();
        node[trans_id].source_queue_length--;
        node[dest_id].recv_ct=node[dest_id].recv_ct+1;

        if(!(node[trans_id].local_queue.empty()))
        {
            node[trans_id].local_queue.front().queue_head_time=time_slot;

        }


    }
}

```

Source-to-Relay Operation

```cpp
#include "my_struc.h"

extern int buffer_bound;
extern int n;
extern int tagged_S;
extern int R_in_number;
extern long lost_number;


void SRtrans(struct Node *node, long time_slot, int trans_id, int recv_id)
{
    int dest_id=0;
    int relay_queue_index=0;
    struct Packet packet;


    /*********if the buffer of relay node is not full!*******************/


    if(node[recv_id].source_queue_length+node[recv_id].sum_relay_queue_length<buffer_bound)
    {

        if(!node[trans_id].local_queue.empty())
        {
            // the traffic pattern is 1<-->2, 3<-->4, ...
            // compute the destination node id
            if(trans_id==n)
                dest_id=1;
            else
                dest_id=trans_id+1;

            if(recv_id==n)
            {
                relay_queue_index=dest_id-1;
            }
            else if(dest_id<recv_id)
            {
                relay_queue_index=dest_id;
            }
            else
            {
                relay_queue_index=dest_id-2;
            }

            packet=node[trans_id].local_queue.front();
            node[recv_id].relay_queue[relay_queue_index].push(packet);
            node[recv_id].sum_relay_queue_length++;

            if(recv_id==tagged_S)
            {
                R_in_number++;
            }

            node[trans id].local queue.pop();
            node[trans_id].source_queue_length--;

        }
    }

    else
    {
        if(!node[trans id].local queue.empty())
        {
            if(trans id==tagged S)
            {
                lost number++;
            }

            if(recv id==tagged S)
            {
                R in number++;
            }

            node[trans id].local queue.pop();
            node[trans id].source queue length--;
        }

    }


    if(!(node[trans_id].local_queue.empty()))
    {
        node[trans_id].local_queue.front().queue_head_time=time_slot;
    }

}

```

Relay-to-Destination Operation

```
#include "my_struc.h"


extern int n;
extern int tagged_D;
extern int R_out_number;
extern float delay;
extern float queuing_delay;
extern float delivery_delay;


void RDtrans(struct Node *node, long time_slot,int trans_id, int recv_id)
{
    int relay_queue_index=0;

    //find the corresponding relay queue in relay node
    if(trans_id==n)
        relay_queue_index=recv_id-1;
    else if(recv_id<trans_id)
        relay_queue_index=recv_id;
    else
        relay_queue_index=recv_id-2;

    //if relay queue has packet
    if(!(node[trans_id].relay_queue[relay_queue_index].empty()))
    {
        node[trans_id].relay_queue[relay_queue_index].front().reception_time=time_slot;

        //we only record a node pair
        if(recv_id==tagged_D)
        {
            struct Packet packet;
            packet=node[trans_id].relay_queue[relay_queue_index].front();


queuing_delay=queuing_delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.queue_head_time-packet
.arrival_time)/(node[recv_id].recv_ct+1);

delivery_delay=delivery_delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.reception_time-packe
t.queue_head_time)/(node[recv_id].recv_ct+1);

delay=delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
ode[recv_id].recv_ct+1);

        }

        //testing the output rate of the relay queue
        if(trans_id==tagged_D)
        {
            R_out_number++;
        }

        node[trans_id].relay_queue[relay_queue_index].pop();
        node[trans_id].sum_relay_queue_length--;
        node[recv_id].recv_ct=node[recv_id].recv_ct+1;
    }
}
```

Random Value Uniformly Distributed on (0,1)

```
#include "macro_def.h"

float ran0_1(long  *idum )
{
    int j;
    long k;
    static  long idum2 =123456789 ;
    static  long iy=0;
    static  long iv[NTAB ];
    float temp ;
    if (*idum <= 0)
    {
        //Initialise.
        if (-(* idum ) < 1) *idum =1;  //Be sure to prevent 'idum' = 0.
        else  *idum = -(* idum );
        idum2 =(* idum );
        for  (j=NTAB +7; j>=0; j--)
        {
            //Load the shuffle table (after 8 warmups).
            k=(* idum )/IQ1;
            *idum =IA1*(*idum -k*IQ1)-k*IR1;
            if (*idum < 0) *idum +=IM1;
            if (j <NTAB )iv[j] = *idum ;
        }
        iy =iv[0];
    }
    k= (*idum )/IQ1; //Start here when not initialising.
    *idum =IA1*(*idum -k*IQ1)-k*IR1; //Compute 'idum=(IA1*idum)' % IM1
    if (*idum < 0) *idum +=IM1;//without overflows by Schrage's method.
    k=idum2 /IQ2;
    idum2 =IA2*(idum2 -k*IQ2)-k*IR2; //Compute 'idum2=(IA2*idum)' % IM2
    if (idum2 < 0)idum2 +=IM2;
    j=iy/NDIV; //Will be in the range 0..NTAB-1.
    iy=iv[j]-idum2 ; //Here 'idum' is shuffles, 'idum' and
    //'idum2' are combined to generate output.
    iv[j] = *idum ;
    if (iy < 1)iy +=IMM1;
    if ((temp =AM*iy) >RNMX) return RNMX; //Because users don't expect endpoint values.
    else  return temp ;
}

```

Probability Generator

```cpp
#include "my_func.h"
#include <iostream>

using namespace std;


extern long seed[1];

int probabilityP(float p)
{
    float sim=ran0_1(seed);
    if(sim<p)
        return 1;
    else
        return 0;
}

```

C++ Simulator for IC-IoT under Buffer-Space Allocation Policy

```
#ifndef MACRO_DEF_H_INCLUDED
#define MACRO_DEF_H_INCLUDED

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define PI 3.141592654



#endif // MACRO_DEF_H_INCLUDED

```

```cpp
#ifndef MY_STRUC_H_INCLUDED
#define MY_STRUC_H_INCLUDED

#include <queue>

using namespace std;

/*************************data structure************************/

struct Packet
{
    int id;              //packet indicator
    int arrival_time ;   //arrival time of this packet
    int queue_head_time;
    int reception_time ; // reception time of this packet
      //the time when this packet arrives its local queue
};

struct Node
{
    int row ; //the row id of this node in a m*m cell-partition network, the cell number C=m*m
    int col ; //the column id of this node

    queue <struct Packet >local_queue ;   //store locally generated packets
    queue <struct Packet > *relay_queue ; //n-2 relay queues to store packets for other traffic flows
    int source_queue_length; //
    int sum_relay_queue_length;


    int arrival_ct ; //total self-generated packets at this node
    int recv_ct ;    //total received packets at this node as a destination
};



struct Cell
{
    int row; //the row id of this cell
    int col; //the column id of this cell

    int *node_in_cell; //recording the node id in this cell;
    int nodenum_of_cell; //recording the node number of this cell;
};

#endif // MY_STRUC_H_INCLUDED

```

```
#ifndef MY_FUNC_H_INCLUDED
#define MY_FUNC_H_INCLUDED

float ran0_1(long  *idum);

int probabilityP(float p);

void update_node_position_IID(int n,int m, struct Node *node);

void update_node_position_RWalk(int n,int m, struct Node *node);

void update_node_position_RWaypoint(int n,int m, struct Node *node);

void collect_nodes_per_cell(int n, int m, struct Node *node);

void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id);

void SRtrans(struct Node *node, long time_slot,int trans_id, int recv_id);

void RDtrans(struct Node *node, long time_slot,int trans_id, int recv_id);

void THROR(struct Node *node, long time_slot);

#endif // MY_FUNC_H_INCLUDED

```

```cpp
/****************Buffer Space Allocation****************/
/****************System Header Files********************/
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <string>

/****************My Header Files**********************/
#include "macro_def.h"
#include "my_func.h"
#include "my_struc.h"


/****************global variables*********************/

long seed[1];

struct Cell **cell;

int source_buffer_bound;
int relay_buffer_bound;

float alpha;

int n;
int m;

int K;


int tagged_S=5;
int tagged_D=6;


/******************test quantities*********************/


int SD_num; //For testing the tagged flow direct transmission opportunity
int SR_num; //For testing the tagged flow S->R transmission opportunity
int RD_num; //For testing the tagged flow R->D transmission opportunity

int S_out_number; //For testing the output opportunity of the local queue
int R_in_number; //For testing the input rate of the relay queue
int R_out_number; //For testing the output rate of the relay queu
long lost_number;

long tagged_S_source_queue_empty;
long tagged_D_source_queue_empty;

long tagged_S_relay_buffer_full;
long tagged_D_relay_buffer_full;

float delay;
float queuing_delay;
float delivery_delay;
/*******************data collection********************/

ofstream ftest;


/*****************Notice! Project Entrance************/
int main()
{
    *seed=-time(0);

    /****************data collection********************/
    ftest.open("buffer space allocation.dat");


    /****************network settings******************/
    n=18;
    m=3;

    source buffer bound=5;
    relay buffer bound=5;

    alpha=0.5;


    cout<<"n: "<<n<<" m: "<<m<<endl;

    /*******************simulation settings***********/
    K=100;
    long time max[K+1];
    time_max[0]=0;
    for(int i=1; i<=K; i++)
    {
        time_max[i]=200000000;
    }


    float lambda[K+1];
    lambda[0]=0;
    for(int i=1; i<=K; i++)
```

```
97      {
98          lambda[i]=0.01*i;
99      }
100
101     long time_slot=-1; //slot 0, slot 1, slot 2, ......
102     int round=1; //for each network setting, how many rounds a simulation has done
103
104
105     /***************building and initializing data structure******************/
106
107     cout<< "System is allocating RAM resource for simulation!!!"<<endl<<endl;
108
109     struct Node *node; //building nodes in the network
110     node=new struct Node[n+1];//node[1],...,node[n]
111
112     cell=new struct Cell *[m]; //
113     for(int i=0; i<m; i++)
114     {
115         cell[i]= new struct Cell[m];
116         for(int j=0; j<m; j++)
117         {
118             cell[i][j].node_in_cell=new int[n+1];
119             cell[i][j].row=i;
120             cell[i][j].col=j;
121         }
122     }
123
124     /***************relay queue for other n-2 flows, the map between node[i].relay_queue[j] and its destined
125     node_id is like this: if, node_id<the current node id index i, then, the current queue index j indicates
126     the destined node_id; else if, node_id>index i, then, j=node_id-2*********************************/
127
128     for(int i=1; i<=n; i++)
129         node[i].relay_queue= new queue<struct Packet>[n-1];  //relay_queue[1], relay_queue[2],..., relay_queue[n-2]
130
131     string mobility_model;
132     mobility_model="IID";
133
134     cout<<"*************************"<<mobility_model<<"*********************************" <<endl;
135     cout<<" n="<<n<<" m="<<m<<" source-buffer-size="<<source_buffer_bound<<"
    relay-buffer-size="<<relay_buffer_bound<<" transmission-ratio="<<alpha<<endl;
136     cout<<"*****************************************************************" <<endl<<endl;
137
138     ftest<<"*************************"<<mobility_model<<"*********************************" <<endl;
139     ftest<<" n="<<n<<" m="<<m<<" source-buffer-size="<<source_buffer_bound<<"
    relay-buffer-size="<<relay_buffer_bound<<" transmission-ratio="<<alpha<<endl;
140     ftest<<"*****************************************************************" <<endl<<endl;
141
142     /************we do simulation as input rate lambda approaches capacity mu*******************/
143     /*********************we just focus on a tagged node pair*************************************/
144
145     for(round=1; round<=K; round++)
146     {
147         //simulation initialization
148
149         cout<<"************new round:"<<round<<" lambda: "<<lambda[round]<<"************"<<endl;
150         //cout<<"************new round:"<<round<<" lambda: "<<lambda<<endl;
151
152         //ftest<<"************new round:"<<round<<" lambda: "<<lambda<<endl;
153
154         ftest<<"************new round:"<<round<<" lambda: "<<lambda[round]<<"************"<<endl;
155
156
157         cout<<" System is initializing simulation status for round: "<<round<<endl;
158
159         //initialize test statistic variables
160
161
162         SD num=0;
163         SR num=0;
164         RD num=0;
165
166
167         S out number=0;
168         R in number=0;
169         R out number=0;
170         lost number=0;
171
172         queuing delay=0;
173         delivery delay=0;
174         delay=0;
175
176         tagged S source queue empty=0;
177         tagged_D_source_queue_empty=0;
178         tagged S relay buffer full=0;
179         tagged_D_relay_buffer_full=0;
180
181         //initialize each node
182         for(int i=1; i<=n; i++)
183         {
184             node[i].row=-1; //node position
185             node[i].col=-1;
186
187             //clear all queues
188             while(!(node[i].local_queue.empty())) //clear local queue
189                 node[i].local_queue.pop();
190
```

```
191             for(int j=0; j<n-1; j++) //clear relay queue
192             {
193                 while(!(node[i].relay_queue[j].empty()))
194                     node[i].relay_queue[j].pop();
195             }
196             node[i].source_queue_length=0;
197             node[i].sum_relay_queue_length=0;
198
199             node[i].arrival_ct=0;
200             node[i].recv_ct=0;
201         }
202
203
204         //initialize time clock
205         time_slot=-1;
206
207         cout<< " Simulation is starting !!! "<<endl;
208
209         //the main simulation body begins here
210         while(time_slot<time_max[round])
211         {
212             time_slot++;
213
214             update_node_position_IID(n, m, node);
215             //update_node_position_RWalk(n, m, node);
216             //update_node_position_RWaypoint(n, m, node);
217
218             collect_nodes_per_cell(n, m, node);
219
220             THROR(node, time_slot);
221
222             //we locally generate packets for all n source nodes, only when time_slot==next_arrival_time
223             for(int i=1; i<=n; i++)
224             {
225
226                 if(probabilityP(lambda[round]))//a new packet arrives in this time slot for node i
227                 {
228                     node[i].arrival_ct++;
229
230                     if(node[i].source_queue_length<source_buffer_bound)
231                     {
232                         struct Packet packet;
233                         packet.id=node[i].arrival_ct;
234                         packet.arrival_time=time_slot;
235                         packet.reception_time=0;
236
237                         if(node[i].local_queue.empty())
238                         {
239                             packet.queue_head_time=time_slot;
240                         }
241                         else
242                         {
243                             packet.queue_head_time=0;
244                         }
245
246                         node[i].local_queue.push(packet);
247                         node[i].source_queue_length++;
248                     }
249                     else
250                     {
251                         if(i==tagged_S)
252                         {
253                             lost number++;
254                         }
255                     }
256                 }
257             }
258
259             if(node[tagged S].source queue length==0)
260             {
261                 tagged S source queue empty++;
262             }
263
264             if(node[tagged D].source queue length==0)
265             {
266                 tagged D source queue empty++;
267             }
268
269             if(node[tagged S].sum relay queue length==relay buffer bound)
270             {
271                 tagged_S_relay_buffer_full++;
272             }
273
274             if(node[tagged D].sum relay queue length==relay buffer bound)
275             {
276                 tagged D relay buffer full++;
277             }
278
279         }
280
281
282
283         cout<<" node "<<tagged_S<<" S-D transmission opportunity: "<<1.0*SD_num/time_slot<<endl;
284
285         cout<<" node "<<tagged_S<<" S-R transmission opportunity: "<<1.0*SR_num/time_slot<<endl;
286
```

```
287             cout<<" node "<<tagged_S<<" R-D transmission opportunity: "<<1.0*RD_num/time_slot<<endl;
288
289             cout<<" node "<<tagged_S<<" generates "<<node[tagged_S].arrival_ct<<" packets in "<<time_slot<<" time
    slots. Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
290
291             cout<<" node "<<tagged_D<<" receives "<<node[tagged_D].recv_ct<<" packets in "<<time_slot<<" time slots.
    Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
292
293             cout<<" node "<<tagged_S<<" loses "<<lost_number<<" packets in "<<time_slot<<" time slots. Packet lost
    rate: "<<1.0*lost_number/time_slot<<endl;
294
295             cout<<" node "<<tagged_S<<" the output opportunity of the local queue is: "<<S_out_number<<"
    "<<1.0*S_out_number/time_slot<<endl;
296
297             cout<<" node "<<tagged_S<<" the input rate of the relay queue is: "<<R_in_number<<"
    "<<1.0*R_in_number/time_slot<<endl;
298
299             cout<<" node "<<tagged_D<<" the output rate of the relay queue is: "<<R_out_number<<"
    "<<1.0*R_out_number/time_slot<<endl;
300
301             cout<<" node "<<tagged_S<<" source queue is empty with probability
    "<<1.0*tagged_S_source_queue_empty/time_slot<<endl;
302
303             cout<<" node "<<tagged_D<<" source queue is empty with probability
    "<<1.0*tagged_D_source_queue_empty/time_slot<<endl;
304
305             cout<<" node "<<tagged_S<<" relay buffer is full with probability
    "<<1.0*tagged_S_relay_buffer_full/time_slot<<endl;
306
307             cout<<" node "<<tagged_D<<" relay buffer is full with probability
    "<<1.0*tagged_D_relay_buffer_full/time_slot<<endl;
308
309             cout<<" node "<<tagged_S<<"'s average packet queuing delay: "<<queuing_delay<<endl;
310
311             cout<<" node "<<tagged_S<<"'s average packet delivery delay: "<<delivery_delay<<endl;
312
313             cout<<" node "<<tagged_S<<"'s average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
314
315             ftest<<" node "<<tagged_S<<" S-D transmission opportunity: "<<1.0*SD_num/time_slot<<endl;
316
317             ftest<<" node "<<tagged_S<<" S-R transmission opportunity: "<<1.0*SR_num/time_slot<<endl;
318
319             ftest<<" node "<<tagged_S<<" R-D transmission opportunity: "<<1.0*RD_num/time_slot<<endl;
320
321             ftest<<"node "<<tagged_S<<" generates "<<node[tagged_S].arrival_ct<<" packets in "<<time_slot<<" time
    slots. Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
322
323             ftest<<"node "<<tagged_D<<" receives "<<node[tagged_D].recv_ct<<" packets in "<<time_slot<<" time slots.
    Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
324
325             ftest<<" node "<<tagged_S<<" loses "<<lost_number<<" packets in "<<time_slot<<" time slots. Packet lost
    rate: "<<1.0*lost_number/time_slot<<endl;
326
327             ftest<<" node "<<tagged S<<" the output opportunity of the local queue is: "<<S out number<<"
    "<<1.0*S_out_number/time_slot<<endl;
328
329             ftest<<" node "<<tagged_S<<" the input rate of the relay queue is: "<<R_in_number<<"
    "<<1.0*R in number/time slot<<endl;
330
331             ftest<<" node "<<tagged_D<<" the output rate of the relay queue is: "<<R_out_number<<"
    "<<1.0*R_out_number/time_slot<<endl;
332
333             ftest<<" node "<<tagged S<<" source queue is empty with probability
    "<<1.0*tagged S source queue empty/time slot<<endl;
334
335             ftest<<" node "<<tagged D<<" source queue is empty with probability
    "<<1.0*tagged D source queue empty/time slot<<endl;
336
337             ftest<<" node "<<tagged S<<" relay buffer is full with probability
    "<<1.0*tagged S relay buffer full/time slot<<endl;
338
339             ftest<<" node "<<tagged D<<" relay buffer is full with probability
    "<<1.0*tagged D relay buffer full/time slot<<endl;
340
341             ftest<<" node "<<tagged S<<"'s average packet queuing delay: "<<queuing delay<<endl;
342
343             ftest<<" node "<<tagged S<<"'s average packet delivery delay: "<<delivery delay<<endl;
344
345             ftest<<" node "<<tagged S<<"'s average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
346         }
347
348         cout<< " System is deleting RAM resources!!!"<<endl<<endl;
349
350         for(int i=0; i<m; i++)
351         {
352             for(int j=0; j<m; j++)
353                 delete []cell[i][j].node_in_cell;
354         }
355
356         for(int i=0; i<m; i++)
357             delete []cell[i];
358         delete []cell;
359
360         for(int i=1; i<=n; i++)
361             delete []node[i].relay_queue;
362         delete []node;
```

```
363
364
365         ftest.close();
366
367         cout << "Simulation is finished!" <<endl;
368         int tmp=0;
369         cin>>tmp;
370
371         return 0;
372     }
373
```

```c
#include "my_func.h"
#include "my_struc.h"

extern long seed[1];


/**
update the position of each node at the beginning of each time slot
according to the i.i.d mobility model
**/

void update_node_position_IID(int n,int m, struct Node *node)
{
    int row =0, col =0;

    for(int i=1;i<=n;i++)
    {
        row=(int)(m*ran0_1(seed)); // random select a row id among 0,1,...,m-1 with equal probability
        col=(int)(m*ran0_1(seed)); // random select a column id among 0,1,...,m-1 with equal probability
        node[i].row=row;
        node[i].col=col;
   }
}


/**
update the position of each node at the beginning of each time slot
according to the random walk mobility model
**/
void update_node_position_RWalk(int n,int m, struct Node *node)
{
    int horizontal_move=0; //-1,0,1
    int vertical_move=0;//-1,0,1

    for(int i=1;i<=n;i++)
    {
        horizontal_move=(int)(3*ran0_1(seed))-1;
        vertical_move=(int)(3*ran0_1(seed))-1;
        node[i].row=(node[i].row+vertical_move+m)%m;
        node[i].col=(node[i].col+horizontal_move+m)%m;
   }
}

/**
update the position of each node at the beginning of each time slot
according to the random way point mobility model
**/
void update_node_position_RWaypoint(int n,int m, struct Node *node)
{
    int v_x=0,v_y=0; //velocity
    int d_x=0,d_y=0; //direction

    for(int i=1;i<=n;i++)
    {
        //determine move direction
        d_x=(int)(2*ran0_1(seed));//0,1
        d_y=(int)(2*ran0_1(seed));//0,1
        if(d_x==0)d_x=-1;
        if(d_y==0)d_y=-1;

        //determine speed
        v_x=(int)(3*ran0_1(seed))+1;//1,2,3
        v_y=(int)(3*ran0_1(seed))+1;//1,2,3

        node[i].row=(node[i].row+d_y*v_y+m)%m;
        node[i].col=(node[i].col+d_x*v_x+m)%m;
    }
}


```

```c
#include "my_struc.h"

extern struct Cell **cell;

/**
collect nodes in each cell
**/
void collect_nodes_per_cell(int n, int m, struct Node *node)
{
    //reset the state of each cell
    for(int i=0; i<m; i++)
    {
        for(int j=0; j<m; j++)
            cell[i][j].nodenum_of_cell=0;
    }

    int r_d=0; //recording the difference between the node row id and the cell row id
    int c_d=0; //recording the difference between the node column id and the cell column id

    int flag=0; // 0: the node is not in this cell 1: the node is the cell

    int index=0; // recording the node is in this cell

    for(int i=1; i<=n; i++)
    {
        flag=0;

        for(int s=0; s<m; s++)
        {
            for(int t=0; t<m; t++)
            {
                r_d=node[i].row-cell[s][t].row;
                c_d=node[i].col-cell[s][t].col;

                if((r_d==0)&&(c_d==0)) //node i is in an active cell
                {
                    index=cell[s][t].nodenum_of_cell;
                    cell[s][t].node_in_cell[index]=i;
                    cell[s][t].nodenum_of_cell++;

                    flag=1;
                    break;
                }
            }

            if(flag==1)
            break;
        }
    }
}
```

```
#include "my_struc.h"
#include "my_func.h"

extern long seed[1];

extern struct Cell **cell;

extern int tagged_S;
extern int tagged_D;

extern int n;
extern int m;


extern int SD_num;
extern int SR_num;
extern int RD_num;
extern int S_out_number;

extern float alpha;

/*********************************************************************************************
                                    Traffic Setting
                  1-->2, 2-->3,...,i-1-->i,...,n-1-->n, n-->1
          with out loss of generality,we focus on a tagged node pair
*********************************************************************************************/

void THROR(struct Node *node, long time_slot)
{


    int dest_id=0; //indicate the direct destination node id
    int trans_id=0; //randomly selected transmitter
    int recv_id=0; //randomly selected receiver
    int index=0;

    int flag=0;

    int nodenum_of_cell=0;



    for(int i=0; i<m; i++)
    {
        for(int j=0; j<m; j++)
        {

            flag=0;

            nodenum of cell=cell[i][j].nodenum of cell;

            if(nodenum_of_cell>=2)
            {
                index=(int)(nodenum of cell*ran0 1(seed));

                trans_id=cell[i][j].node_in_cell[index];

                if(trans id==n)
                    dest id=1;
                else
                    dest_id=trans_id+1;


                for(int tmp=0; tmp<nodenum of cell; tmp++)
                {
                    if(cell[i][j].node in cell[tmp]==dest id)
                    {
                        SDtrans(node,time slot,trans id,dest id);

                        if(trans id==tagged S)
                        {
                            S out number++;
                            SD num++;
                        }

                        flag=1;
                        break;
                    }
                }

                if(flag==0)
                {
                    do{
                        index=(int)(nodenum of cell*ran0 1(seed));
                        recv_id=cell[i][j].node_in_cell[index];
                    }while(recv id==trans id);

/***************transmission scheduling: with probability alpha, do S->R, with probability 1-alpha, do
R->D*************/

                    if(probabilityP(alpha))                         //do S->R
                    {
                        SRtrans(node,time_slot,trans_id,recv_id);
                        if(trans_id==tagged_S)
                        {
                            SR_num++;
```

```
                                S_out_number++;
                            }

                        }


                        else //do R->D
                        {
                            RDtrans(node,time_slot,trans_id,recv_id);
                            if(trans_id==tagged_S)
                                RD_num++;
                        }


                    }

                }
            }
        }

}

```

```cpp
#include "my_struc.h"



extern int tagged_S;

extern float delay;
extern float queuing_delay;
extern float delivery_delay;


void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id)
{
    if(!(node[trans_id].local_queue.empty())) //if the local queue has packet
    {
        node[trans_id].local_queue.front().reception_time=time_slot; //recording the reception time of this packet

        if(trans_id==tagged_S) //without loss of generality, we focus on a tagged SD pair
        {
            struct Packet packet;
            packet=node[trans_id].local_queue.front();









queuing_delay=queuing_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.queue_head_time-packet.arrival_time)/(node[dest_id].recv_ct+1);

delivery_delay=delivery_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packet.queue_head_time)/(node[dest_id].recv_ct+1);

delay=delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(node[dest_id].recv_ct+1);



        }


        node[trans_id].local_queue.pop();
        node[trans_id].source_queue_length--;
        node[dest_id].recv_ct=node[dest_id].recv_ct+1;

        if(!(node[trans_id].local_queue.empty()))
        {
            node[trans_id].local_queue.front().queue_head_time=time_slot;

        }


    }
}

```

```cpp
#include "my_struc.h"

extern int relay_buffer_bound;
extern int n;
extern int tagged_S;
extern int R_in_number;
extern long lost_number;


void SRtrans(struct Node *node, long time_slot, int trans_id, int recv_id)
{
    int dest_id=0;
    int relay_queue_index=0;
    struct Packet packet;


    /*********if the local queue has packet and the relay queue is not full!*******************/


    if(node[recv_id].sum_relay_queue_length<relay_buffer_bound)
    {

        if(!node[trans_id].local_queue.empty())
        {
            // the traffic pattern is 1<-->2, 3<-->4, ...
            // compute the destination node id
            if(trans_id==n)
                dest_id=1;
            else
                dest_id=trans_id+1;

            if(recv_id==n)
            {
                relay_queue_index=dest_id-1;
            }
            else if(dest_id<recv_id)
            {
                relay_queue_index=dest_id;
            }
            else
            {
                relay_queue_index=dest_id-2;
            }

            packet=node[trans_id].local_queue.front();
            node[recv_id].relay_queue[relay_queue_index].push(packet);
            node[recv_id].sum_relay_queue_length++;

            if(recv_id==tagged_S)
            {
                R_in_number++;
            }

            node[trans id].local queue.pop();
            node[trans_id].source_queue_length--;

        }
    }

    else
    {
        if(!node[trans id].local queue.empty())
        {
            if(trans id==tagged S)
            {
                lost number++;
            }

            if(recv id==tagged S)
            {
                R in number++;
            }

            node[trans id].local queue.pop();
            node[trans id].source queue length--;
        }

    }


    if(!(node[trans_id].local_queue.empty()))
    {
        node[trans_id].local_queue.front().queue_head_time=time_slot;
    }

}

```

```cpp
#include "my_struc.h"


extern int n;
extern int tagged_D;
extern int R_out_number;
extern float delay;
extern float queuing_delay;
extern float delivery_delay;


void RDtrans(struct Node *node, long time_slot,int trans_id, int recv_id)
{
    int relay_queue_index=0;

    //find the corresponding relay queue in relay node
    if(trans_id==n)
        relay_queue_index=recv_id-1;
    else if(recv_id<trans_id)
        relay_queue_index=recv_id;
    else
        relay_queue_index=recv_id-2;

    //if relay queue has packet
    if(!(node[trans_id].relay_queue[relay_queue_index].empty()))
    {
        node[trans_id].relay_queue[relay_queue_index].front().reception_time=time_slot;

        //we only record a node pair
        if(recv_id==tagged_D)
        {
            struct Packet packet;
            packet=node[trans_id].relay_queue[relay_queue_index].front();


queuing_delay=queuing_delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.queue_head_time-packet.arrival_time)/(node[recv_id].recv_ct+1);

delivery_delay=delivery_delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.reception_time-packet.queue_head_time)/(node[recv_id].recv_ct+1);

delay=delay*(1.0*node[recv_id].recv_ct/(node[recv_id].recv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(node[recv_id].recv_ct+1);

        }

        //testing the output rate of the relay queue
        if(trans_id==tagged_D)
        {
            R_out_number++;
        }

        node[trans_id].relay_queue[relay_queue_index].pop();
        node[trans_id].sum_relay_queue_length--;
        node[recv_id].recv_ct=node[recv_id].recv_ct+1;
    }
}
```

```
#include "macro_def.h"

float ran0_1(long  *idum  )
{
    int j;
    long k;
    static  long idum2  =123456789 ;
    static  long iy=0;
    static  long iv[NTAB ];
    float temp ;
    if  (*idum  <= 0)
        {
         //Initialise.
         if  (-(* idum  ) < 1) *idum  =1;  //Be sure to prevent 'idum' = 0.
         else  *idum  = -(* idum  );
         idum2  =(* idum  );
         for  (j=NTAB +7;j>=0;j--)
             {
                 //Load the shuffle table (after 8 warmups).
                 k=(* idum  )/IQ1;
                 *idum  =IA1*(*idum  -k*IQ1)-k*IR1;
                 if  (* idum  < 0) *idum  += IM1 ;
                 if  (j <NTAB ) iv[j] = *idum  ;
             }
         iy =iv[0];
        }
        k=  (* idum  )/IQ1; //Start here when not initialising.
        *idum  =IA1*(*idum  -k*IQ1)-k*IR1; //Compute 'idum=(IA1*idum)' % IM1
        if  (* idum  < 0) *idum  += IM1 ;//without overflows by Schrage's method.
        k=idum2  /IQ2 ;
        idum2  =IA2*(idum2  -k*IQ2 )-k*IR2; //Compute 'idum2=(IA2*idum)' % IM2
        if  (idum2  < 0)idum2  += IM2;
        j=iy/NDIV; //Will be in the range 0..NTAB-1.
        iy=iv[j]-idum2  ; //Here 'idum' is shuffles, 'idum' and
        //'idum2' are combined to generate output.
        iv[j] = *idum  ;
        if  (iy < 1)iy += IMM1;
        if  ((temp =AM*iy) >RNMX) return RNMX; //Because users don't expect endpoint values.
        else  return temp ;
}

```

```cpp
#include "my_func.h"
#include <iostream>

using namespace std;


extern long seed[1];

int probabilityP(float p)
{
    float sim=ran0_1(seed);
    if(sim<p)
        return 1;
    else
        return 0;
}

```