```
#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

```

```
#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 n*n 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 H2HR(struct Node *node, long time_slot);

#endif // MY_FUNC_H_INCLUDED

```

```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
#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"
#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 H2HR(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++;
                        }

                    }


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


                }

            }
        }
    }

}
```

```
#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;
```

```
84          }
85
86      }
87
```

```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;
extern int S_out_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((trans_id==tagged_S)&&(node[recv_id].sum_relay_queue_length<relay_buffer_bound))
        S_out_number++;

    if(!node[trans_id].local_queue.empty() && node[recv_id].sum_relay_queue_length<relay_buffer_bound)
    {
        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--;

        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;
    }
}


```

```
/*****************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 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; //For testing the probability that relay queue if full
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("L_F_R_F_FB.dat");


    /****************network settings*******************/
    n=72;
    m=6;

    source_buffer_bound=5;
    relay_buffer_bound=5;

    alpha=0.5;

    float mu;
    mu=0.0177;

    //float mus=0.153;

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

```

```
85      /*******************simulation settings***********/
86      long time_max[10]= {0,100000000,100000000,100000000,100000000,100000000,200000000,
    200000000,20000000,20000000}; //simulation runs for time_max slots
87
88      long time_slot=-1; //slot_0, slot_1, slot_2, ......
89      int round=1; //for each network setting, how many rounds a simulation has done
90
91      //system load: rho=lambda/mu
92      float rho[10]= {0,0.002,0.005,0.01,0.015,0.02,0.05,0.1,0.2,0.5};
93
94      float lambda;
95      lambda=0;
96
97
98      /***************building and initializing data structure*******************/
99
100     cout<< "System is allocating RAM resource for simulation!!!"<<endl<<endl;
101
102     struct Node *node; //building nodes in the network
103     node=new struct Node[n+1];//node[1],...,node[n]
104
105     cell=new struct Cell *[m]; //
106     for(int i=0; i<m; i++)
107     {
108         cell[i]= new struct Cell[m];
109         for(int j=0; j<m; j++)
110         {
111             cell[i][j].node_in_cell=new int[n+1];
112             cell[i][j].row=i;
113             cell[i][j].col=j;
114         }
115     }
116
117     /***************relay queue for other n-2 flows, the map between
    node[i].relay_queue[j] and its destined
118     node id is like this: if, node id<the current node id index i, then, the current queue
    index j indicates
119     the destined node id; else if, node id>index i, then,
    j=node id-2***********************************/
120
121     for(int i=1; i<=n; i++)
122         node[i].relay_queue= new queue<struct Packet>[n-1];  //relay_queue[1],
    relay_queue[2],..., relay_queue[n-2]
123
124     string mobility_model;
125     mobility_model="IID";
126
127     cout<<"*************************"<<mobility_model<<
    "*********************************" <<endl;
128     cout<<" n="<<n<<" m="<<m<<" source-buffer-size="<<source_buffer_bound<<"
    relay-buffer-size="<<relay_buffer_bound<<" transmission-ratio="<<alpha<<" Feedback"<<"
    capacity="<<mu<<endl;
129     cout<<"************************************************************" <<endl<<endl;
130
131     ftest<<"*************************"<<mobility_model<<
    "*********************************" <<endl;
132     ftest<<" n="<<n<<" m="<<m<<" source-buffer-size="<<source_buffer_bound<<"
    relay-buffer-size="<<relay_buffer_bound<<" transmission-ratio="<<alpha<<" Feedback"<<"
    capacity="<<mu<<endl;
133     ftest<<"************************************************************" <<endl<<endl;
134
135     /***********we do simulation as input rate lambda approaches capacity
    mu*********************/
136     /*******************we just focus on a tagged node
    pair************************************/
137
138     for(round=1; round<=9; round++)
139     {
140         //simulation initialization
141         lambda=rho[round]; //input rate for this simulation round
142
143         cout<<"***********new round:"<<round<<" rho="<<rho[round]<<" lambda: "<<lambda<<
    "************"<<endl;
144         //cout<<"***********new round:"<<round<<" lamda: "<<lambda<<endl;
145
146         //ftest<<"***********new round:"<<round<<" lamda: "<<lambda<<endl;
147
148         ftest<<"***********new round:"<<round<<" rho="<<rho[round]<<" lambda: "<<lambda<<
    "************"<<endl;
149
150
151         cout<<" System is initializing simulation status for round: "<<round<<endl;
152
153         //initialize test statistic variables
```

```
154
155
156             SD_num=0;
157             SR_num=0;
158             RD_num=0;
159
160
161             S_out_number=0;
162             R_in_number=0;
163             R_out_number=0;
164             lost_number=0;
165
166             queuing_delay=0;
167             delivery_delay=0;
168             delay=0;
169
170             tagged_S_source_queue_empty=0;
171             tagged_D_source_queue_empty=0;
172             tagged_S_relay_buffer_full=0;
173             tagged_D_relay_buffer_full=0;
174
175             //initialize each node
176             for(int i=1; i<=n; i++)
177             {
178                 node[i].row=-1; //node position
179                 node[i].col=-1;
180
181                 //clear all queues
182                 while(!(node[i].local_queue.empty())) //clear local queue
183                     node[i].local_queue.pop();
184
185                 for(int j=0; j<n-1; j++) //clear relay queue
186                 {
187                     while(!(node[i].relay_queue[j].empty()))
188                         node[i].relay_queue[j].pop();
189                 }
190                 node[i].source_queue_length=0;
191                 node[i].sum_relay_queue_length=0;
192
193                 node[i].arrival_ct=0;
194                 node[i].recv_ct=0;
195             }
196
197
198             //initialize time clock
199             time_slot=-1;
200
201             cout<< " Simulation is starting !!! "<<endl;
202
203             //the main simulation body begins here
204             while(time_slot<time_max[round])
205             {
206                 time_slot++;
207
208                 update_node_position_IID(n, m, node);
209                 //update_node_position_RWalk(n, m, node);
210                 //update_node_position_RWaypoint(n, m, node);
211
212                 collect_nodes_per_cell(n, m, node);
213
214                 H2HR(node, time_slot);
215
216                 //we locally generate packets for all n source nodes, only when
    time_slot==next_arrival_time
217                 for(int i=1; i<=n; i++)
218                 {
219
220                     if(probabilityP(lambda))//a new packet arrives in this time slot for node i
221                     {
222                         node[i].arrival_ct++;
223
224                         if(node[i].source_queue_length<source_buffer_bound)
225                         {
226                             struct Packet packet;
227                             packet.id=node[i].arrival_ct;
228                             packet.arrival_time=time_slot;
229                             packet.reception_time=0;
230
231                             if(node[i].local_queue.empty())
232                             {
233                                 packet.queue_head_time=time_slot;
234                             }
235                             else
236                             {
```

```
                                packet.queue_head_time=0;
                            }

                            node[i].local_queue.push(packet);
                            node[i].source_queue_length++;
                        }
                        else
                        {
                            if(i==tagged_S)
                            {
                                lost_number++;
                            }
                        }
                    }
                }

                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].sum_relay_queue_length==relay_buffer_bound)
                {
                    tagged_S_relay_buffer_full++;
                }

                if(node[tagged_D].sum_relay_queue_length==relay_buffer_bound)
                {
                    tagged_D_relay_buffer_full++;
                }

            }



            cout<<" node "<<tagged_S<<" S-D transmission opportunity: "<<1.0*SD_num/time_slot<<
        endl;

            cout<<" node "<<tagged_S<<" S-R transmission opportunity: "<<1.0*SR_num/time_slot<<
        endl;

            cout<<" node "<<tagged_S<<" R-D transmission opportunity: "<<1.0*RD_num/time_slot<<
        endl;

            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;

            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;

            cout<<" node "<<tagged_S<<" loses "<<lost_number<<" packets in "<<time_slot<<" time
        slots. Packet lost rate: "<<1.0*lost_number/time_slot<<endl;

            cout<<" node "<<tagged_S<<" the output opportunity of the local queue is: "<<
        S_out_number<<" "<<1.0*S_out_number/time_slot<<endl;

            cout<<" node "<<tagged_S<<" the input rate of the relay queue is: "<<R_in_number<<"
        "<<1.0*R_in_number/time_slot<<endl;

            cout<<" node "<<tagged_D<<" the output rate of the relay queue is: "<<R_out_number
        <<" "<<1.0*R_out_number/time_slot<<endl;

            cout<<" node "<<tagged_S<<" source queue is empty with probability "<<1.0*
        tagged_S_source_queue_empty/time_slot<<endl;

            cout<<" node "<<tagged_D<<" source queue is empty with probability "<<1.0*
        tagged_D_source_queue_empty/time_slot<<endl;

            cout<<" node "<<tagged_S<<" relay queue is full with probability "<<1.0*
        tagged_S_relay_buffer_full/time_slot<<endl;

            cout<<" node "<<tagged_D<<" relay queue is full with probability "<<1.0*
        tagged_D_relay_buffer_full/time_slot<<endl;

            cout<<" node "<<tagged_S<<"'s average packet queuing delay: "<<queuing_delay<<endl;

            cout<<" node "<<tagged_S<<"'s average packet delivery delay: "<<delivery_delay<<endl;

            cout<<" node "<<tagged_S<<"'s average packet end-to-end delay: "<<delay<<endl<<endl
```

```
    <<endl;
308
309
310
311
312
313
314         ftest<<" node "<<tagged_S<<" S-D transmission opportunity: "<<1.0*SD_num/time_slot
    <<endl;
315
316         ftest<<" node "<<tagged_S<<" S-R transmission opportunity: "<<1.0*SR_num/time_slot
    <<endl;
317
318         ftest<<" node "<<tagged_S<<" R-D transmission opportunity: "<<1.0*RD_num/time_slot
    <<endl;
319
320         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;
321
322         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;
323
324         ftest<<" node "<<tagged_S<<" loses "<<lost_number<<" packets in "<<time_slot<<"
    time slots. Packet lost rate: "<<1.0*lost_number/time_slot<<endl;
325
326         ftest<<" node "<<tagged_S<<" the output opportunity of the local queue is: "<<
    S_out_number<<" "<<1.0*S_out_number/time_slot<<endl;
327
328         ftest<<" node "<<tagged_S<<" the input rate of the relay queue is: "<<R_in_number<<
    " "<<1.0*R_in_number/time_slot<<endl;
329
330         ftest<<" node "<<tagged_D<<" the output rate of the relay queue is: "<<R_out_number
    <<" "<<1.0*R_out_number/time_slot<<endl;
331
332         ftest<<" node "<<tagged_S<<" source queue is empty with probability "<<1.0*
    tagged_S_source_queue_empty/time_slot<<endl;
333
334         ftest<<" node "<<tagged_D<<" source queue is empty with probability "<<1.0*
    tagged_D_source_queue_empty/time_slot<<endl;
335
336         ftest<<" node "<<tagged_S<<" relay queue is full with probability "<<1.0*
    tagged_S_relay_buffer_full/time_slot<<endl;
337
338         ftest<<" node "<<tagged_D<<" relay queue is full with probability "<<1.0*
    tagged_D_relay_buffer_full/time_slot<<endl;
339
340         ftest<<" node "<<tagged_S<<"'s average packet queuing delay: "<<queuing_delay<<endl;
341
342         ftest<<" node "<<tagged_S<<"'s average packet delivery delay: "<<delivery_delay<<
    endl;
343
344         ftest<<" node "<<tagged_S<<"'s average packet end-to-end delay: "<<delay<<endl<<endl
    <<endl;
345     }
346
347     cout<< " System is deleting RAM resources!!!"<<endl<<endl;
348
349     for(int i=0; i<m; i++)
350     {
351         for(int j=0; j<m; j++)
352             delete []cell[i][j].node_in_cell;
353     }
354
355     for(int i=0; i<m; i++)
356         delete []cell[i];
357     delete []cell;
358
359     for(int i=1; i<=n; i++)
360         delete []node[i].relay_queue;
361     delete []node;
362
363
364     ftest.close();
365
366     cout << "Simulation is finished!" <<endl;
367     int tmp=0;
368     cin>>tmp;
369
370     return 0;
371 }
372
```