

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

Macro Define

```
1  #ifndef MACRO_DEF_H_INCLUDED
2  #define MACRO_DEF_H_INCLUDED
3
4  #define IM1 2147483563
5  #define IM2 2147483399
6  #define AM (1.0/IM1)
7  #define IMM1 (IM1-1)
8  #define IA1 40014
9  #define IA2 40692
10 #define IQ1 53668
11 #define IQ2 52774
12 #define IR1 12211
13 #define IR2 3791
14 #define NTAB 32
15 #define NDIV (1+IMM1/NTAB)
16 #define EPS 1.2e-7
17 #define RNMX (1.0-EPS)
18 #define PI 3.141592654
19
20
21
22 #endif // MACRO_DEF_H_INCLUDED
23
```

Structure Define

```
1  #ifndef MY_STRUC_H_INCLUDED
2  #define MY_STRUC_H_INCLUDED
3
4  #include <queue>
5
6  using namespace std;
7
8  /*****data structure*****/
9
10 struct Packet
11 {
12     int id; //packet indicator
13     int arrival_time; //arrival time of this packet
14     int queue_head_time;
15     int reception_time; // reception time of this packet
16     //the time when this packet arrives its local queue
17 };
18
19 struct Node
20 {
21     int row; //the row id of this node in a m*m cell-partition network, the cell number C=m*m
22     int col; //the column id of this node
23
24     queue<struct Packet> local_queue; //store locally generated packets
25     queue<struct Packet> *relay_queue; //n-2 relay queues to store packets for other traffic flows
26     int source_queue_length; //
27     int sum_relay_queue_length; //
28
29
30     int arrival_ct; //total self-generated packets at this node
31     int recv_ct; //total received packets at this node as a destination
32 };
33
34
35
36 struct Cell
37 {
38     int row; //the row id of this cell
39     int col; //the column id of this cell
40
41     int *node_in_cell; //recording the node id in this cell;
42     int nodenum_of_cell; //recording the node number of this cell;
43 };
44
45 #endif // MY_STRUC_H_INCLUDED
46
```

Function Declaration

```
1  #ifndef MY_FUNC_H_INCLUDED
2  #define MY_FUNC_H_INCLUDED
3
4  float ran0_1(long *idum);
5
6  int probabilityP(float p);
7
8  void update_node_position_IID(int n, int m, struct Node *node);
9
10 void update_node_position_RWalk(int n, int m, struct Node *node);
11
12 void update_node_position_RWaypoint(int n, int m, struct Node *node);
13
14 void collect_nodes_per_cell(int n, int m, struct Node *node);
15
16 void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id);
17
18 void SRtrans(struct Node *node, long time_slot, int trans_id, int recv_id);
19
20 void RDtrans(struct Node *node, long time_slot, int trans_id, int recv_id);
21
22 void THRROR(struct Node *node, long time_slot);
23
24 #endif // MY_FUNC_H_INCLUDED
25
```

Main Function

```
1  /*****Buffer Space Sharing*****/
2
3  /*****System Header Files*****/
4  #include <iostream>
5  #include <fstream>
6  #include <cmath>
7  #include <cstdlib>
8  #include <ctime>
9  #include <string>
10
11 /*****My Header Files*****/
12 #include "macro_def.h"
13 #include "my_func.h"
14 #include "my_struct.h"
15
16
17 /*****global variables*****/
18
19 long seed [1];
20
21 struct Cell **cell;
22
23 int buffer_bound;
24
25 float alpha;
26
27 int n;
28 int m;
29
30 int K;
31
32 int tagged_S=5;
33 int tagged_D=6;
34
35
36 /*****test quantities*****/
37
38
39 int SD_num; //For testing the tagged flow direct transmission opportunity
40 int SR_num; //For testing the tagged flow S->R transmission opportunity
41 int RD_num; //For testing the tagged flow R->D transmission opportunity
42
43 int S_out_number; //For testing the output opportunity of the local queue
44 int R_in_number; //For testing the input rate of the relay queue
45 int R_out_number; //For testing the output rate of the relay queue
46 long lost_number;
47
48 long tagged_S_source_queue_empty;
49 long tagged_D_source_queue_empty;
50
51 long tagged_S_buffer_empty;
52 long tagged_D_buffer_empty;
53
54 long tagged_S_buffer_full;
55 long tagged_D_buffer_full;
56
57 float delay;
58 float queuing delay;
59 float delivery delay;
60 /*****data collection*****/
61
62 ofstream ftest;
63
64
65 /*****Notice! Project Entrance*****/
66 int main()
67 {
68     *seed =-time(0);
69
70     /*****data collection*****/
71     ftest.open("data.dat");
72
73
74     /*****network settings*****/
75     n=18;
76     m=3;
77
78     buffer_bound=10;
79
80     alpha=0.5;
81
82     //float mus=0.153;
83
84     cout<<"n: "<<n<<" m: "<<m<<endl;
85
86     /*****simulation settings*****/
87     K=100;
88     long time_max[K+1];
89     time_max[0]=0;
90     for(int i=1; i<=K; i++)
91     {
92         time_max[i]=20000000;
93     }
94
95
96     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<<" lambda: "<<lambda<<endl;
153
154         //fics1<<"*****new round:"<<round<<" lambda: "<<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         SD num=0;
164         SR num=0;
165         RD num=0;
166
167
168         S out number=0;
169         R in number=0;
170         R out number=0;
171         lost number=0;
172
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

```

```

193     node[i].local_queue.pop();
194
195     for(int j=0; j<n-1; j++) //clear relay queue
196     {
197         while(!(node[i].relay_queue[j].empty()))
198             node[i].relay_queue[j].pop();
199     }
200     node[i].source_queue_length=0;
201     node[i].sum_relay_queue_length=0;
202
203     node[i].arrival_ct=0;
204     node[i].recv_ct=0;
205 }
206
207
208 //initialize time clock
209 time_slot=-1;
210
211 cout<< " Simulation is starting !!! "<<endl;
212
213 //the main simulation body begins here
214 while(time_slot<time_max[round])
215 {
216     time_slot++;
217
218     update_node_position_IID(n, m, node);
219     //update_node_position_RWalk(n, m, node);
220     //update_node_position_RWaypoint(n, m, node);
221
222     collect_nodes_per_cell(n, m, node);
223
224     THROR(node, time_slot); //包在时隙未到达
225
226     //we locally generate packets for all n source nodes, only when time_slot==next arrival time
227     for(int i=1; i<=n; i++)
228     {
229
230         if(probabilityP(lambda[round]))//a new packet arrives in this time slot for node i
231         {
232             node[i].arrival_ct++;
233
234             if(node[i].source_queue_length+node[i].sum_relay_queue_length<buffer_bound)
235             {
236                 struct Packet packet;
237                 packet.id=node[i].arrival_ct;
238                 packet.arrival_time=time_slot;
239                 packet.reception_time=0;
240
241                 if(node[i].local_queue.empty())
242                 {
243                     packet.queue_head_time=time_slot;
244                 }
245                 else
246                 {
247                     packet.queue_head_time=0;
248                 }
249
250                 node[i].local_queue.push(packet);
251                 node[i].source_queue_length++;
252             }
253             else
254             {
255                 if(i==tagged_S)
256                 {
257                     lost_number++;
258                 }
259             }
260         }
261     }
262
263     //U2HR(node, time_slot); //包在时隙初到达
264
265     if(node[tagged_S].source_queue_length==0)
266     {
267         tagged_S_source_queue_empty++;
268     }
269
270     if(node[tagged_D].source_queue_length==0)
271     {
272         tagged_D_source_queue_empty++;
273     }
274
275     if(node[tagged_S].source_queue_length+node[tagged_S].sum_relay_queue_length==0)
276     {
277         tagged_S_buffer_empty++;
278     }
279
280     if(node[tagged_D].source_queue_length+node[tagged_D].sum_relay_queue_length==0)
281     {
282         tagged_D_buffer_empty++;
283     }
284
285     if(node[tagged_S].source_queue_length+node[tagged_S].sum_relay_queue_length==buffer_bound)
286     {
287         tagged_S_buffer_full++;
288     }

```

```

289         if (node[tagged_D].source_queue_length+node[tagged_D].sum_relay_queue_length==buffer_bound)
290         {
291             tagged_D_buffer_full++;
292         }
293     }
294 }
295
296
297     cout<<" Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
298
299     cout<<" Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
300
301     cout<<" Average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
302
303
304     ftest<<" Input rate: "<<1.0*node[tagged_S].arrival_ct/time_slot<<endl;
305
306     ftest<<" Throughput rate: "<<1.0*node[tagged_D].recv_ct/time_slot<<endl;
307
308     ftest<<" Average packet end-to-end delay: "<<delay<<endl<<endl<<endl;
309
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

```
1  #include "my_func.h"
2  #include "my_struct.h"
3
4  extern long seed[1];
5
6
7  /**
8  update the position of each node at the beginning of each time slot
9  according to the i.i.d mobility model
10 */
11
12 void update_node_position_IID(int n, int m, struct Node *node)
13 {
14     int row = 0, col = 0;
15
16     for(int i=1; i<=n; i++)
17     {
18         row=(int)(m*ran0_1(seed)); // random select a row id among 0,1,...,m-1 with equal probability
19         col=(int)(m*ran0_1(seed)); // random select a column id among 0,1,...,m-1 with equal probability
20         node[i].row=row;
21         node[i].col=col;
22     }
23 }
24
25
26 /**
27 update the position of each node at the beginning of each time slot
28 according to the random walk mobility model
29 */
30 void update_node_position_RWalk(int n, int m, struct Node *node)
31 {
32     int horizontal_move=0; //-1, 0, 1
33     int vertical_move=0; //-1, 0, 1
34
35     for(int i=1; i<=n; i++)
36     {
37         horizontal_move=(int)(3*ran0_1(seed))-1;
38         vertical_move=(int)(3*ran0_1(seed))-1;
39         node[i].row=(node[i].row+vertical_move+m)%m;
40         node[i].col=(node[i].col+horizontal_move+m)%m;
41     }
42 }
43
44 /**
45 update the position of each node at the beginning of each time slot
46 according to the random way point mobility model
47 */
48 void update_node_position_RWaypoint(int n, int m, struct Node *node)
49 {
50     int v_x=0, v_y=0; //velocity
51     int d_x=0, d_y=0; //direction
52
53     for(int i=1; i<=n; i++)
54     {
55         //determine move direction
56         d_x=(int)(2*ran0_1(seed)); //0, 1
57         d_y=(int)(2*ran0_1(seed)); //0, 1
58         if(d_x==0) d_x=-1;
59         if(d_y==0) d_y=-1;
60
61         //determine speed
62         v_x=(int)(3*ran0_1(seed))+1; //1, 2, 3
63         v_y=(int)(3*ran0_1(seed))+1; //1, 2, 3
64
65         node[i].row=(node[i].row+d_y*v_y+m)%m;
66         node[i].col=(node[i].col+d_x*v_x+m)%m;
67     }
68 }
69
70
```

Collect Device Information

```
1  #include "my_struct.h"
2
3  extern struct Cell **cell ;
4
5  /**
6  collect nodes in each cell
7  **/
8  void collect_nodes_per_cell(int n, int m, struct Node *node )
9  {
10     //reset the state of each cell
11     for(int i=0; i<m; i++)
12     {
13         for(int j=0; j<m; j++)
14             cell [i][j]. nodedum_of_cell=0;
15     }
16
17     int r_d=0; //recording the difference between the node row id and the cell row id
18     int c_d=0; //recording the difference between the node column id and the cell column id
19
20     int flag=0; // 0: the node is not in this cell 1: the node is the cell
21
22     int index=0; // recording the node is in this cell
23
24     for(int i=1; i<=n; i++)
25     {
26         flag=0;
27
28         for(int s=0; s<m; s++)
29         {
30             for(int t=0; t<m; t++)
31             {
32                 r_d=node [i]. row-cell [s][t]. row ;
33                 c_d=node [i]. col-cell [s][t]. col;
34
35                 if((r_d==0)&&(c_d==0)) //node i is in an active cell
36                 {
37                     index=cell [s][t]. nodedum_of_cell;
38                     cell [s][t]. node_in_cell[index]=i;
39                     cell [s][t]. nodedum_of_cell++;
40
41                     flag=1;
42                     break;
43                 }
44             }
45
46             if(flag==1)
47                 break;
48         }
49     }
50 }
51
```

Two-hop Relaying Opportunistic Routing Algorithm

```

1  #include "my_struct.h"
2  #include "my_func.h"
3
4  extern long seed[1];
5
6  extern struct Cell **cell;
7
8  extern int tagged_S;
9  extern int tagged_D;
10
11 extern int n;
12 extern int m;
13
14
15 extern int SD_num;
16 extern int SR_num;
17 extern int RD_num;
18 extern int S_out_number;
19
20 extern float alpha;
21
22 /*****
23                                     Traffic Setting
24                                     1→2, 2→3,...,i-1→i,...,n-1→n, n→1
25                                     with out loss of generality,we focus on a tagged node pair
26 *****/
27
28 void THROR(struct Node *node, long time_slot)
29 {
30
31
32     int dest_id=0; //indicate the direct destination node id
33     int trans_id=0; //randomly selected transmitter
34     int rcv_id=0; //randomly selected receiver
35     int index=0;
36
37     int flag=0;
38
39     int nodenum_of_cell=0;
40
41
42
43     for(int i=0; i<m; i++)
44     {
45         for(int j=0; j<m; j++)
46         {
47
48             flag=0;
49
50             nodenum of cell=cell[i][j].nodenum of cell;
51
52             if(nodenum_of_cell>=2)
53             {
54                 index=(int)(nodenum of cell*ran0_1(seed));
55
56                 trans_id=cell[i][j].node_in_cell[index];
57
58                 if(trans_id==n)
59                     dest_id=1;
60                 else
61                     dest_id=trans_id+1;
62
63
64                 for(int tmp=0; tmp<nodenum of cell; tmp++)
65                 {
66                     if(cell[i][j].node_in_cell[tmp]==dest_id)
67                     {
68                         SDtrans(node,time_slot,trans_id,dest_id);
69
70                         if(trans_id==tagged_S)
71                         {
72                             S_out_number++;
73                             SD_num++;
74                         }
75
76                         flag=1;
77                         break;
78                     }
79                 }
80
81                 if(flag==0)
82                 {
83                     do
84                     {
85                         index=(int)(nodenum_of_cell*ran0_1(seed));
86                         rcv_id=cell[i][j].node_in_cell[index];
87                     }
88                     while(rcv_id==trans_id);
89
90                     /*****transmission scheduling: with probability alpha, do S→R, with probability
91                     1-alpha, do R→D*****/
92
93                     if(probabilityP(alpha)) //do S→R
94                     {
95                         SRtrans(node,time_slot,trans_id,rcv_id);
96                         if(trans_id==tagged_S)

```

```
96         {
97             SR_num++;
98             S_out_number++;
99         }
100     }
101 }
102
103
104 else //do R->D
105 {
106     RDtrans(node, time_slot, trans_id, recv_id);
107     if(trans_id==tagged_S)
108         RD_num++;
109 }
110
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
```

Source-to-Destination Operation

```
1  #include "my_struct.h"
2
3  extern int tagged_S;
4  extern float delay;
5  extern float queuing_delay;
6  extern float delivery_delay;
7
8  void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id)
9  {
10     if(!(node[trans_id].local_queue.empty())) //if the local queue has packet
11     {
12         node[trans_id].local_queue.front().reception_time=time_slot; //recording the reception time of this packet
13
14         if(trans_id==tagged_S) //without loss of generality, we focus on a tagged SD pair
15         {
16             struct Packet packet;
17             packet=node[trans_id].local_queue.front();
18
19             queuing_delay=queuing_delay*(1.0*node[dest_id].rcv_ct/(node[dest_id].rcv_ct+1))+1.0*(packet.queue_head_time-packet
20             .arrival_time)/(node[dest_id].rcv_ct+1);
21
22             delivery_delay=delivery_delay*(1.0*node[dest_id].rcv_ct/(node[dest_id].rcv_ct+1))+1.0*(packet.reception_time-packet
23             t.queue_head_time)/(node[dest_id].rcv_ct+1);
24
25             delay=delay*(1.0*node[dest_id].rcv_ct/(node[dest_id].rcv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
26             ode[dest_id].rcv_ct+1);
27
28         }
29
30         node[trans_id].local_queue.pop();
31         node[trans_id].source_queue_length--;
32         node[dest_id].rcv_ct=node[dest_id].rcv_ct+1;
33
34         if(!(node[trans_id].local_queue.empty()))
35         {
36             node[trans_id].local_queue.front().queue_head_time=time_slot;
37         }
38     }
39 }
```

Source-to-Relay Operation

```
1  #include "my_struct.h"
2
3  extern int buffer_bound;
4  extern int n;
5  extern int tagged_S;
6  extern int R_in_number;
7  extern long lost_number;
8
9
10 void SRtrans(struct Node *node, long time_slot, int trans_id, int recv_id)
11 {
12     int dest_id=0;
13     int relay_queue_index=0;
14     struct Packet packet;
15
16
17     /*****if the buffer of relay node is not full!*****/
18
19
20     if(node[recv_id].source_queue_length+node[recv_id].sum_relay_queue_length<buffer_bound)
21     {
22
23         if(!node[trans_id].local_queue.empty())
24         {
25             // the traffic pattern is 1<-->2, 3<-->4, ...
26             // compute the destination node id
27             if(trans_id==n)
28                 dest_id=1;
29             else
30                 dest_id=trans_id+1;
31
32             if(recv_id==n)
33             {
34                 relay_queue_index=dest_id-1;
35             }
36             else if(dest_id<recv_id)
37             {
38                 relay_queue_index=dest_id;
39             }
40             else
41             {
42                 relay_queue_index=dest_id-2;
43             }
44
45             packet=node[trans_id].local_queue.front();
46             node[recv_id].relay_queue[relay_queue_index].push(packet);
47             node[recv_id].sum_relay_queue_length++;
48
49             if(recv_id==tagged_S)
50             {
51                 R_in_number++;
52             }
53
54             node[trans_id].local_queue.pop();
55             node[trans_id].source_queue_length--;
56
57         }
58     }
59
60     else
61     {
62         if(!node[trans_id].local_queue.empty())
63         {
64             if(trans_id==tagged_S)
65             {
66                 lost_number++;
67             }
68
69             if(recv_id==tagged_S)
70             {
71                 R_in_number++;
72             }
73
74             node[trans_id].local_queue.pop();
75             node[trans_id].source_queue_length--;
76         }
77     }
78
79
80     if(!(node[trans_id].local_queue.empty()))
81     {
82         node[trans_id].local_queue.front().queue_head_time=time_slot;
83     }
84 }
85
86 }
87
```

Relay-to-Destination Operation

```
1  #include "my_struct.h"
2
3
4  extern int n;
5  extern int tagged_D;
6  extern int R_out_number;
7  extern float delay;
8  extern float queuing_delay;
9  extern float delivery_delay;
10
11
12 void RDtrans(struct Node *node, long time_slot, int trans_id, int rcv_id)
13 {
14     int relay_queue_index=0;
15
16     //find the corresponding relay queue in relay node
17     if(trans_id==n)
18         relay_queue_index=rcv_id-1;
19     else if(rcv_id<trans_id)
20         relay_queue_index=rcv_id;
21     else
22         relay_queue_index=rcv_id-2;
23
24     //if relay queue has packet
25     if(!(node[trans_id].relay_queue[relay_queue_index].empty()))
26     {
27         node[trans_id].relay_queue[relay_queue_index].front().reception_time=time_slot;
28
29         //we only record a node pair
30         if(rcv_id==tagged_D)
31         {
32             struct Packet packet;
33             packet=node[trans_id].relay_queue[relay_queue_index].front();
34
35             queuing_delay=queuing_delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.queue_head_time-packet
36             .arrival_time)/(node[rcv_id].rcv_ct+1);
37
38             delivery_delay=delivery_delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.reception_time-packe
39             t.queue_head_time)/(node[rcv_id].rcv_ct+1);
40
41             delay=delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
42             ode[rcv_id].rcv_ct+1);
43         }
44
45         //testing the output rate of the relay queue
46         if(trans_id==tagged_D)
47         {
48             R_out_number++;
49         }
50
51         node[trans_id].relay_queue[relay_queue_index].pop();
52         node[trans_id].sum_relay_queue_length--;
53         node[rcv_id].rcv_ct=node[rcv_id].rcv_ct+1;
54     }
55 }
```

Random Value Uniformly Distributed on (0,1)

```

1  #include "macro_def.h"
2
3  float ran0_1(long *idum )
4  {
5      int j;
6      long k;
7      static long idum2 =123456789 ;
8      static long iy=0;
9      static long iv[NTAB ];
10     float temp ;
11     if (*idum <= 0)
12     {
13         //Initialise.
14         if (-(* idum ) < 1) *idum =1; //Be sure to prevent 'idum' = 0.
15         else *idum = -(* idum );
16         idum2 =(* idum );
17         for (j=NTAB +7; j>=0; j--)
18         {
19             //Load the shuffle table (after 8 warmups).
20             k=(* idum )/IQ1;
21             *idum =IA1*( *idum -k*IQ1)-k*IR1;
22             if (*idum < 0) *idum +=IM1;
23             if (j < NTAB ) iv[j] = *idum ;
24         }
25         iy =iv[0];
26     }
27     k= (*idum )/IQ1; //Start here when not initialising.
28     *idum =IA1*( *idum -k*IQ1)-k*IR1; //Compute 'idum=(IA1*idum)' % IM1
29     if (*idum < 0) *idum +=IM1; //without overflows by Schrage's method.
30     k=idum2 /IQ2;
31     idum2 =IA2*(idum2 -k*IQ2)-k*IR2; //Compute 'idum2=(IA2*idum)' % IM2
32     if (idum2 < 0) idum2 +=IM2;
33     j=iy/NDIV; //Will be in the range 0..NTAB-1.
34     iy=iv[j]-idum2 ; //Here 'idum' is shuffles, 'idum' and
35     // 'idum2' are combined to generate output.
36     iv[j] = *idum ;
37     if (iy < 1) iy +=IMM1;
38     if ((temp =AM*iy) >RNMx) return RNMx; //Because users don't expect endpoint values.
39     else return temp ;
40 }
41

```


Probability Generator

```
1  #include "my_func.h"
2  #include <iostream>
3
4  using namespace std;
5
6
7  extern long seed[1];
8
9  int probabilityP(float p)
10 {
11     float sim=ran0_1(seed);
12     if(sim<p)
13         return 1;
14     else
15         return 0;
16 }
17
```

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

```
1  #ifndef MACRO_DEF_H_INCLUDED
2  #define MACRO_DEF_H_INCLUDED
3
4  #define IM1 2147483563
5  #define IM2 2147483399
6  #define AM (1.0/IM1)
7  #define IMM1 (IM1-1)
8  #define IA1 40014
9  #define IA2 40692
10 #define IQ1 53668
11 #define IQ2 52774
12 #define IR1 12211
13 #define IR2 3791
14 #define NTAB 32
15 #define NDIV (1+IMM1/NTAB)
16 #define EPS 1.2e-7
17 #define RNMX (1.0-EPS)
18 #define PI 3.141592654
19
20
21
22 #endif // MACRO_DEF_H_INCLUDED
23
```

```

1  #ifndef MY_STRUC_H_INCLUDED
2  #define MY_STRUC_H_INCLUDED
3
4  #include <queue>
5
6  using namespace std;
7
8  /*****data structure*****/
9
10 struct Packet
11 {
12     int id; //packet indicator
13     int arrival_time ; //arrival time of this packet
14     int queue_head_time;
15     int reception_time ; // reception time of this packet
16     //the time when this packet arrives its local queue
17 };
18
19 struct Node
20 {
21     int row ; //the row id of this node in a m*m cell-partition network, the cell number C=m*m
22     int col ; //the column id of this node
23
24     queue <struct Packet > local_queue ; //store locally generated packets
25     queue <struct Packet > *relay_queue ; //n-2 relay queues to store packets for other traffic flows
26     int source_queue_length; //
27     int sum_relay_queue_length;
28
29
30     int arrival_ct ; //total self-generated packets at this node
31     int recv_ct ; //total received packets at this node as a destination
32 };
33
34
35
36 struct Cell
37 {
38     int row; //the row id of this cell
39     int col; //the column id of this cell
40
41     int *node_in_cell; //recording the node id in this cell;
42     int nodenum_of_cell; //recording the node number of this cell;
43 };
44
45 #endif // MY_STRUC_H_INCLUDED
46

```

```

1  #ifndef MY_FUNC_H_INCLUDED
2  #define MY_FUNC_H_INCLUDED
3
4  float ran0_1(long *idum);
5
6  int probabilityP(float p);
7
8  void update_node_position_IID(int n, int m, struct Node *node);
9
10 void update_node_position_RWalk(int n, int m, struct Node *node);
11
12 void update_node_position_RWaypoint(int n, int m, struct Node *node);
13
14 void collect_nodes_per_cell(int n, int m, struct Node *node);
15
16 void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id);
17
18 void SRtrans(struct Node *node, long time_slot, int trans_id, int rcv_id);
19
20 void RDtrans(struct Node *node, long time_slot, int trans_id, int rcv_id);
21
22 void THRROR(struct Node *node, long time_slot);
23
24 #endif // MY_FUNC_H_INCLUDED
25

```

```

1  /*****Buffer Space Allocation*****/
2  /*****System Header Files*****/
3  #include <iostream>
4  #include <fstream>
5  #include <cmath>
6  #include <cstdlib>
7  #include <ctime>
8  #include <string>
9
10 /*****My Header Files*****/
11 #include "macro_def.h"
12 #include "my_func.h"
13 #include "my_struct.h"
14
15 /*****global variables*****/
16
17 long seed[1];
18
19 struct Cell **cell;
20
21 int source_buffer_bound;
22 int relay_buffer_bound;
23
24 float alpha;
25
26 int n;
27 int m;
28
29 int K;
30
31 int tagged_S=5;
32 int tagged_D=6;
33
34 /*****test quantities*****/
35
36 int SD_num; //For testing the tagged flow direct transmission opportunity
37 int SR_num; //For testing the tagged flow S->R transmission opportunity
38 int RD_num; //For testing the tagged flow R->D transmission opportunity
39
40 int S_out_number; //For testing the output opportunity of the local queue
41 int R_in_number; //For testing the input rate of the relay queue
42 int R_out_number; //For testing the output rate of the relay queue
43 long lost_number;
44
45 long tagged_S_source_queue_empty;
46 long tagged_D_source_queue_empty;
47
48 long tagged_S_relay_buffer_full;
49 long tagged_D_relay_buffer_full;
50
51 float delay;
52 float queuing_delay;
53 float delivery_delay;
54 /*****data collection*****/
55
56 ofstream ftest;
57
58 /*****Notice! Project Entrance*****/
59 int main()
60 {
61     *seed=-time(0);
62
63     /*****data collection*****/
64     ftest.open("buffer space allocation.dat");
65
66     /*****network settings*****/
67     n=18;
68     m=3;
69
70     source_buffer_bound=5;
71     relay_buffer_bound=5;
72
73     alpha=0.5;
74
75     cout<<"n: "<<n<<" m: "<<m<<endl;
76
77     /*****simulation settings*****/
78     K=100;
79     long time_max[K+1];
80     time_max[0]=0;
81     for(int i=1; i<=K; i++)
82     {
83         time_max[i]=200000000;
84     }
85
86     float lambda[K+1];
87     lambda[0]=0;
88     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[i] 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 << "
136     relay-buffer-size=" << relay_buffer_bound << " transmission-ratio=" << alpha << endl;
137     cout<< "*****" << endl<< endl;
138     ftest<< "*****" << mobility_model << "*****" << endl;
139     ftest<< " n=" << n << " m=" << m << " source-buffer-size=" << source_buffer_bound << "
140     relay-buffer-size=" << relay_buffer_bound << " transmission-ratio=" << alpha << endl;
141     ftest<< "*****" << endl<< endl;
142
143     /*****we do simulation as input rate lambda approaches capacity mu*****/
144     /*****we just focus on a tagged node pair*****/
145
146     for(round=1; round<=K; round++)
147     {
148         //simulation initialization
149
150         cout<< "*****new round:" << round << " lambda: " << lambda[round] << "*****" << endl;
151         //cout<< "*****new round:" << round << " lambda: " << lambda << endl;
152         //fgetc<< "*****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         SD num=0;
162         SR num=0;
163         RD num=0;
164
165
166         S out number=0;
167         R in number=0;
168         R out number=0;
169         lost number=0;
170
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 //initialize time clock
204 time_slot=-1;
205
206 cout<< " Simulation is starting !!! "<<endl;
207
208 //the main simulation body begins here
209 while(time_slot<time_max[round])
210 {
211     time_slot++;
212
213     update_node_position_IID(n, m, node);
214     //update_node_position_RWalk(n, m, node);
215     //update_node_position_RWaypoint(n, m, node);
216
217     collect_nodes_per_cell(n, m, node);
218
219     THRROR(node, time_slot);
220
221     //we locally generate packets for all n source nodes, only when time_slot==next arrival time
222     for(int i=1; i<=n; i++)
223     {
224         if(probabilityP(lambda[round]))//a new packet arrives in this time slot for node i
225         {
226             node[i].arrival_ct++;
227
228             if(node[i].source_queue_length<source_buffer_bound)
229             {
230                 struct Packet packet;
231                 packet.id=node[i].arrival_ct;
232                 packet.arrival_time=time_slot;
233                 packet.reception_time=0;
234
235                 if(node[i].local_queue.empty())
236                 {
237                     packet.queue_head_time=time_slot;
238                 }
239                 else
240                 {
241                     packet.queue_head_time=0;
242                 }
243
244                 node[i].local_queue.push(packet);
245                 node[i].source_queue_length++;
246             }
247             else
248             {
249                 if(i==tagged_S)
250                 {
251                     lost number++;
252                 }
253             }
254         }
255     }
256
257     if(node[tagged_S].source_queue_length==0)
258     {
259         tagged_S_source_queue_empty++;
260     }
261
262     if(node[tagged_D].source_queue_length==0)
263     {
264         tagged_D_source_queue_empty++;
265     }
266
267     if(node[tagged_S].sum_relay_queue_length==relay_buffer_bound)
268     {
269         tagged_S_relay_buffer_full++;
270     }
271
272     if(node[tagged_D].sum_relay_queue_length==relay_buffer_bound)
273     {
274         tagged_D_relay_buffer_full++;
275     }
276
277 }
278
279
280
281
282 cout<<" node "<<tagged_S<<" S-D transmission opportunity: "<<1.0*SD_num/time_slot<<endl;
283
284 cout<<" node "<<tagged_S<<" S-R transmission opportunity: "<<1.0*SR_num/time_slot<<endl;
285
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
```

```

1  #include "my_func.h"
2  #include "my_struct.h"
3
4  extern long seed[1];
5
6
7  /**
8  update the position of each node at the beginning of each time slot
9  according to the i.i.d mobility model
10 */
11
12 void update_node_position_IID(int n, int m, struct Node *node)
13 {
14     int row =0, col =0;
15
16     for(int i=1; i<=n; i++)
17     {
18         row=(int)(m*ran0_1(seed)); // random select a row id among 0,1,...,m-1 with equal probability
19         col=(int)(m*ran0_1(seed)); // random select a column id among 0,1,...,m-1 with equal probability
20         node[i].row=row;
21         node[i].col=col;
22     }
23 }
24
25
26 /**
27 update the position of each node at the beginning of each time slot
28 according to the random walk mobility model
29 */
30 void update_node_position_RWalk(int n, int m, struct Node *node)
31 {
32     int horizontal_move=0; //-1, 0, 1
33     int vertical_move=0; //-1, 0, 1
34
35     for(int i=1; i<=n; i++)
36     {
37         horizontal_move=(int)(3*ran0_1(seed))-1;
38         vertical_move=(int)(3*ran0_1(seed))-1;
39         node[i].row=(node[i].row+vertical_move+m)%m;
40         node[i].col=(node[i].col+horizontal_move+m)%m;
41     }
42 }
43
44 /**
45 update the position of each node at the beginning of each time slot
46 according to the random way point mobility model
47 */
48 void update_node_position_RWaypoint(int n, int m, struct Node *node)
49 {
50     int v_x=0, v_y=0; //velocity
51     int d_x=0, d_y=0; //direction
52
53     for(int i=1; i<=n; i++)
54     {
55         //determine move direction
56         d_x=(int)(2*ran0_1(seed)); //0, 1
57         d_y=(int)(2*ran0_1(seed)); //0, 1
58         if(d_x==0) d_x=-1;
59         if(d_y==0) d_y=-1;
60
61         //determine speed
62         v_x=(int)(3*ran0_1(seed))+1; //1, 2, 3
63         v_y=(int)(3*ran0_1(seed))+1; //1, 2, 3
64
65         node[i].row=(node[i].row+d_y*v_y+m)%m;
66         node[i].col=(node[i].col+d_x*v_x+m)%m;
67     }
68 }
69
70

```

```

1  #include "my_struct.h"
2
3  extern struct Cell **cell;
4
5  /**
6  collect nodes in each cell
7  **/
8  void collect_nodes_per_cell(int n, int m, struct Node *node)
9  {
10     //reset the state of each cell
11     for(int i=0; i<m; i++)
12     {
13         for(int j=0; j<m; j++)
14             cell[i][j].nodenum_of_cell=0;
15     }
16
17     int r_d=0; //recording the difference between the node row id and the cell row id
18     int c_d=0; //recording the difference between the node column id and the cell column id
19
20     int flag=0; // 0: the node is not in this cell 1: the node is the cell
21
22     int index=0; // recording the node is in this cell
23
24     for(int i=1; i<=n; i++)
25     {
26         flag=0;
27
28         for(int s=0; s<m; s++)
29         {
30             for(int t=0; t<m; t++)
31             {
32                 r_d=node[i].row-cell[s][t].row;
33                 c_d=node[i].col-cell[s][t].col;
34
35                 if((r_d==0)&&(c_d==0)) //node i is in an active cell
36                 {
37                     index=cell[s][t].nodenum_of_cell;
38                     cell[s][t].node_in_cell[index]=i;
39                     cell[s][t].nodenum_of_cell++;
40
41                     flag=1;
42                     break;
43                 }
44             }
45
46             if(flag==1)
47                 break;
48         }
49     }
50 }
51

```

```

1  #include "my_struct.h"
2  #include "my_func.h"
3
4  extern long seed[1];
5
6  extern struct Cell **cell;
7
8  extern int tagged_S;
9  extern int tagged_D;
10
11 extern int n;
12 extern int m;
13
14
15 extern int SD_num;
16 extern int SR_num;
17 extern int RD_num;
18 extern int S_out_number;
19
20 extern float alpha;
21
22 /*****
23                                     Traffic Setting
24                                     1→2, 2→3,...,i-1→i,...,n-1→n, n→1
25                                     with out loss of generality,we focus on a tagged node pair
26 *****/
27
28 void THROR(struct Node *node, long time_slot)
29 {
30
31     int dest_id=0; //indicate the direct destination node id
32     int trans_id=0; //randomly selected transmitter
33     int rcv_id=0; //randomly selected receiver
34     int index=0;
35
36     int flag=0;
37
38     int nodenum_of_cell=0;
39
40
41
42
43     for(int i=0; i<m; i++)
44     {
45         for(int j=0; j<m; j++)
46         {
47
48             flag=0;
49
50             nodenum of cell=cell[i][j].nodenum of cell;
51
52             if(nodenum_of_cell>=2)
53             {
54                 index=(int)(nodenum of cell*ran0 1(seed));
55
56                 trans_id=cell[i][j].node_in_cell[index];
57
58                 if(trans_id==n)
59                     dest_id=1;
60                 else
61                     dest_id=trans_id+1;
62
63
64                 for(int tmp=0; tmp<nodenum of cell; tmp++)
65                 {
66                     if(cell[i][j].node_in_cell[tmp]==dest_id)
67                     {
68                         SDtrans(node,time_slot,trans_id,dest_id);
69
70                         if(trans_id==tagged_S)
71                         {
72                             S_out_number++;
73                             SD_num++;
74                         }
75
76                         flag=1;
77                         break;
78                     }
79                 }
80
81                 if(flag==0)
82                 {
83                     do{
84                         index=(int)(nodenum of cell*ran0 1(seed));
85                         rcv_id=cell[i][j].node_in_cell[index];
86                     }while(rcv_id==trans_id);
87
88 /*****transmission scheduling: with probability alpha, do S→R, with probability 1-alpha, do
89 R→D*****/
90
91                     if(probabilityP(alpha)) //do S→R
92                     {
93                         SRtrans(node,time_slot,trans_id,rcv_id);
94                         if(trans_id==tagged_S)
95                         {
96                             SR_num++;

```

```
96         S_out_number++;
97     }
98
99 }
100
101
102 else //do R->D
103 {
104     RDtrans(node, time_slot, trans_id, recv_id);
105     if(trans_id==tagged_S)
106         RD_num++;
107 }
108
109
110 }
111
112 }
113
114 }
115
116 }
117
```

```

1  #include "my_struct.h"
2
3
4
5  extern int tagged_S;
6
7  extern float delay;
8  extern float queuing_delay;
9  extern float delivery_delay;
10
11
12 void SDtrans(struct Node *node, long time_slot, int trans_id, int dest_id)
13 {
14     if(!(node[trans_id].local_queue.empty())) //if the local queue has packet
15     {
16         node[trans_id].local_queue.front().reception_time=time_slot; //recording the reception time of this packet
17
18         if(trans_id==tagged_S) //without loss of generality, we focus on a tagged SD pair
19         {
20             struct Packet packet;
21             packet=node[trans_id].local_queue.front();
22
23
24
25
26
27
28         queuing_delay=queuing_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.queue_head_time-packet
29         .arrival_time)/(node[dest_id].recv_ct+1);
30
31         delivery_delay=delivery_delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packet
32         t.queue_head_time)/(node[dest_id].recv_ct+1);
33
34         delay=delay*(1.0*node[dest_id].recv_ct/(node[dest_id].recv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
35         ode[dest_id].recv_ct+1);
36
37     }
38
39     node[trans_id].local_queue.pop();
40     node[trans_id].source_queue_length--;
41     node[dest_id].recv_ct=node[dest_id].recv_ct+1;
42
43     if(!(node[trans_id].local_queue.empty()))
44     {
45         node[trans_id].local_queue.front().queue_head_time=time_slot;
46     }
47 }
48 }
49

```

```

1  #include "my_struct.h"
2
3  extern int relay_buffer_bound;
4  extern int n;
5  extern int tagged_S;
6  extern int R_in_number;
7  extern long lost_number;
8
9
10 void SRtrans(struct Node *node, long time_slot, int trans_id, int recv_id)
11 {
12     int dest_id=0;
13     int relay_queue_index=0;
14     struct Packet packet;
15
16
17     /*****if the local queue has packet and the relay queue is not full!*****/
18
19
20     if(node[recv_id].sum_relay_queue_length<relay_buffer_bound)
21     {
22
23         if(!node[trans_id].local_queue.empty())
24         {
25             // the traffic pattern is 1<-->2, 3<-->4, ...
26             // compute the destination node id
27             if(trans_id==n)
28                 dest_id=1;
29             else
30                 dest_id=trans_id+1;
31
32             if(recv_id==n)
33             {
34                 relay_queue_index=dest_id-1;
35             }
36             else if(dest_id<recv_id)
37             {
38                 relay_queue_index=dest_id;
39             }
40             else
41             {
42                 relay_queue_index=dest_id-2;
43             }
44
45             packet=node[trans_id].local_queue.front();
46             node[recv_id].relay_queue[relay_queue_index].push(packet);
47             node[recv_id].sum_relay_queue_length++;
48
49             if(recv_id==tagged_S)
50             {
51                 R_in_number++;
52             }
53
54             node[trans_id].local_queue.pop();
55             node[trans_id].source_queue_length--;
56
57         }
58     }
59
60     else
61     {
62         if(!node[trans_id].local_queue.empty())
63         {
64             if(trans_id==tagged_S)
65             {
66                 lost_number++;
67             }
68
69             if(recv_id==tagged_S)
70             {
71                 R_in_number++;
72             }
73
74             node[trans_id].local_queue.pop();
75             node[trans_id].source_queue_length--;
76         }
77     }
78
79
80     if(!(node[trans_id].local_queue.empty()))
81     {
82         node[trans_id].local_queue.front().queue_head_time=time_slot;
83     }
84
85
86 }
87

```



```

1  #include "my_struct.h"
2
3
4  extern int n;
5  extern int tagged_D;
6  extern int R_out_number;
7  extern float delay;
8  extern float queuing_delay;
9  extern float delivery_delay;
10
11
12 void RDtrans(struct Node *node, long time_slot, int trans_id, int rcv_id)
13 {
14     int relay_queue_index=0;
15
16     //find the corresponding relay queue in relay node
17     if(trans_id==n)
18         relay_queue_index=rcv_id-1;
19     else if(rcv_id<trans_id)
20         relay_queue_index=rcv_id;
21     else
22         relay_queue_index=rcv_id-2;
23
24     //if relay queue has packet
25     if(!(node[trans_id].relay_queue[relay_queue_index].empty()))
26     {
27         node[trans_id].relay_queue[relay_queue_index].front().reception_time=time_slot;
28
29         //we only record a node pair
30         if(rcv_id==tagged_D)
31         {
32             struct Packet packet;
33             packet=node[trans_id].relay_queue[relay_queue_index].front();
34
35             queuing_delay=queuing_delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.queue_head_time-packet
36             .arrival_time)/(node[rcv_id].rcv_ct+1);
37
38             delivery_delay=delivery_delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.reception_time-packe
39             t.queue_head_time)/(node[rcv_id].rcv_ct+1);
40
41             delay=delay*(1.0*node[rcv_id].rcv_ct/(node[rcv_id].rcv_ct+1))+1.0*(packet.reception_time-packet.arrival_time)/(n
42             ode[rcv_id].rcv_ct+1);
43         }
44
45         //testing the output rate of the relay queue
46         if(trans_id==tagged_D)
47         {
48             R_out_number++;
49         }
50
51         node[trans_id].relay_queue[relay_queue_index].pop();
52         node[trans_id].sum_relay_queue_length--;
53         node[rcv_id].rcv_ct=node[rcv_id].rcv_ct+1;
54     }
55 }

```

```

1  #include "macro_def.h"
2
3  float ran0_1(long *idum )
4  {
5      int j;
6      long k;
7      static long idum2  =123456789 ;
8      static long iy=0;
9      static long iv[NTAB ];
10     float temp ;
11     if (*idum <= 0)
12     {
13         //Initialise.
14         if (-(*idum ) < 1) *idum  =1; //Be sure to prevent 'idum' = 0.
15         else *idum  = -(*idum );
16         idum2  =(*idum );
17         for (j=NTAB +7; j>=0; j--)
18         {
19             //Load the shuffle table (after 8 warmups).
20             k=(*idum )/IQ1;
21             *idum  =IA1*(*idum  -k*IQ1)-k*IR1;
22             if (*idum  < 0) *idum  += IM1 ;
23             if (j < NTAB ) iv[j] = *idum  ;
24         }
25         iy =iv[0];
26     }
27     k= (*idum )/IQ1; //Start here when not initialising.
28     *idum  =IA1*(*idum  -k*IQ1)-k*IR1; //Compute 'idum=(IA1*idum)' % IM1
29     if (*idum  < 0) *idum  += IM1 ;//without overflows by Schrage's method.
30     k=idum2  /IQ2 ;
31     idum2  =IA2*(*idum2  -k*IQ2 )-k*IR2; //Compute 'idum2=(IA2*idum)' % IM2
32     if (idum2  < 0) idum2  += IM2;
33     j=iy/NDIV; //Will be in the range 0..NTAB-1.
34     iy=iv[j]-idum2  ; //Here 'idum' is shuffles, 'idum' and
35     // 'idum2' are combined to generate output.
36     iv[j] = *idum  ;
37     if (iy < 1) iy += IMM1;
38     if ((temp =AM*iy) >RNMx) return RNMx; //Because users don't expect endpoint values.
39     else return temp ;
40 }
41

```

```
1  #include "my_func.h"
2  #include <iostream>
3
4  using namespace std;
5
6
7  extern long seed[1];
8
9  int probabilityP(float p)
10 {
11     float sim=ran0_1(seed);
12     if(sim<p)
13         return 1;
14     else
15         return 0;
16 }
17
```