```
/*****************************************************************
Name                :     Larry Bush
Due Date            :     December 3, 2002
email               :     bushl2@rpi.edu
Student ID          :     660 220 742

Assignment          :     Course Project (Hot Potato Routing Simulation)
Class               :     CSCI-4966/6965 -
                                Parallel and Distributed Simulation
Instructor          :     Chris Carothers

File                :     routing.c
Executables         :     routing-par and routing-seq
Location            :     ~bushl2/ross-2.1/routing/routing-grape
*****************************************************************

This is a simulation analysis of the algorithm
presented in "Routing without Flow Control," by Busch,
Herlihy and Wattenhoffer, 2001, [1].
The hot-potato routing algorithm simulation is written for
Rensselaer's Optimistic Simulation system (ROSS).
The simulation uses a novel reverse computation approach
to efficiently and optimistically parallelize the system
simulation.

*/

/*  Include File(s)  */
#include <ross.h>

/*****************************************************************/
/*****************************************************************/
/********                                                 ********/
/********        Global Variables and Definitions         ********/
/********                                                 ********/
/*****************************************************************/
/*****************************************************************/

#define TRACER_CODE_ONoff
#define ROUTERS_CAN_INJECT_MULTIPLE_PACKETSoff

int NUM_KP_X = 8;
int NUM_KP_Y = 8;

int NUM_ROUTERS_X = 4;
int NUM_ROUTERS_Y = 4;

#define TIME_STEP 100
int SIMULATION_DURATION = 40000;

enum bool { false, true };

double probability_q = 24;
double probability_p = 16;
int probability_i = 1;
int absorb_sleeping_packet = 1;
```

```
enum bool one_shot = false;



int NumLps;
int NumLpsRT;


/*****************************************************************/
/*****************************************************************/
/********                                                 ********/
/********                Msg_Data Struct                  ********/
/********                                                 ********/
/*****************************************************************/
/*****************************************************************/

/*  Event List to include in the struct.  */
enum events { ARRIVE, ROUTE, HEARTBEAT, PACKET_INJECTION_APPLICATION };


/*  Direction List to include in the struct.  */
enum directions { NORTH, EAST, SOUTH, WEST, NO_DIRECTION };

enum priorities { RUNNING_HOME, RUNNING, TURNING,  EXCITED, ACTIVE, SLEEPING,
                  PLACE_HOLDER, INJECTING };

enum link_status { NOT_FREE, FREE };

/*  The struct.
This stores data passed from this event to the next event.*/
typedef struct {
      enum events event_type;
      enum directions direction;
      enum priorities priority;

      int destination_row;
      int destination_column;

      double leave_time;
      double Saved_total_delivery_time;

      int Saved_injection_wait_time_steps;
      int Saved_worst_case_injection_wait_time_steps;
      int distance;
} Msg_Data;


/*****************************************************************/
/*****************************************************************/
/********                                                 ********/
/********                Router State Struct              ********/
/********                                                 ********/
/*****************************************************************/
/*****************************************************************/
/*
This struct holds all the variables for each router.
```

```
*/

typedef struct {

      enum link_status link[4];

      double total_delivery_time;
      int total_distance_traversed;

      int TotalDelivered;
      int injection_wait_time_steps;
      int   total_injection_wait_time_steps;
      int   worst_case_injection_wait_time_steps;
      int total_packets_injected;

} Router_State;

/* Declare and initialize stat counters. */
double Mean_total_delivery_time = 0;
double Mean_total_distance_traversed  = 0;
int global_TotalDelivered = 0;
int   global_total_injection_wait_time_steps = 0;
int   global_worst_case_injection_wait_time_steps = 0;
int global_total_packets_injected = 0;

void Router_StartUp(Router_State *, tw_lp *);
void Router_EventHandler(Router_State *, tw_bf *, Msg_Data *, tw_lp *);
void Router_RC_EventHandler(Router_State *, tw_bf *, Msg_Data *, tw_lp *);
void Router_Statistics_CollectStats(Router_State *, tw_lp *);

#define Router 1  /* This is just a number to identify the LP */

/* Set up array of logical processors.
   The order of the function names within the struct allows
   ROSS to know what the main functions for this simulation are.  */
tw_lptype Router_lps[] = {
      {
            Router, sizeof(Router_State),
                  (init_f) Router_StartUp,
                  (event_f) Router_EventHandler,
                  (revent_f) Router_RC_EventHandler,
                  (final_f) Router_Statistics_CollectStats,
                  (statecp_f) NULL
      },
      { 0 },
};
/*************************************************************/
/*************************************************************/
/********************                     ********************/
/********************          MAIN       ********************/
/********************                     ********************/
/*************************************************************/
/*************************************************************//**/
int main(int argc, char * argv[]) {

      int number_of_processors;
```

```
int number_of_lps = NUM_ROUTERS_X * NUM_ROUTERS_Y;
int             x, y, xkp, ykp;

int num_lps_per_kp;
int kp_per_proc;
int number_of_kps;

int i = 0;
tw_lp *lp;
tw_kp *kp;
g_tw_gvt_interval = 16;
g_tw_mblock = 16;


if (argc == 7) {

  /*****************************************************************************/
  /*  Read in user input from command line and assign variables accordingly. */
  /*****************************************************************************/

  NUM_ROUTERS_X = atoi((argv)[1]);

  NUM_ROUTERS_Y = NUM_ROUTERS_X;
  number_of_lps = NUM_ROUTERS_X * NUM_ROUTERS_Y;

  NumLps = NUM_ROUTERS_X * NUM_ROUTERS_Y;
  NumLpsRT = NUM_ROUTERS_X;

  number_of_processors = atoi(argv[2]);
  g_tw_npe = number_of_processors;

  SIMULATION_DURATION = atoi(argv[3]);

  g_tw_ts_end = SIMULATION_DURATION;

  probability_i = atoi(argv[4]);

  NUM_KP_X = atoi(argv[6]);
  NUM_KP_Y = atoi(argv[6]);


  if(probability_i == 0) { one_shot = true; }
  /* static (not dynamic) packet insertion. */

  absorb_sleeping_packet = atoi(argv[5]);

} else {
  /*******************************************************/
  /*  Read in user input and assign variables accordingly. */
  /*******************************************************/
  printf("\n\nWhat is the number of rows in the routing matrix? ");

  scanf("%d", &NUM_ROUTERS_X);

  NUM_ROUTERS_Y = NUM_ROUTERS_X;
  number_of_lps = NUM_ROUTERS_X * NUM_ROUTERS_Y;
```

```
  NumLps = NUM_ROUTERS_X * NUM_ROUTERS_Y;
  NumLpsRT = NUM_ROUTERS_X;

  printf("\n\nWhat is the number of processors? ");
  scanf("%d", &number_of_processors);
  g_tw_npe = number_of_processors;

  printf("\n\nWhat is the simulation duration? ");
  scanf("%d", &SIMULATION_DURATION);

  g_tw_ts_end = SIMULATION_DURATION;

  printf("\n\nHow many routers should inject packets in percentage (i.e. 0, 50,
          100)? \n\n");
  scanf("%d", &probability_i);

  if(probability_i == 0) { one_shot = true; }
  /* static (not dynamic) packet insertion. */

  printf("Should a router absorb a sleeping packet? ");
  scanf("%d", &absorb_sleeping_packet);

  printf("Number of KPs : ");
  scanf("%d", &NUM_KP_X);
  NUM_KP_Y  =  NUM_KP_X;

}

/*********************************************************/
/************   Configuration Print Out     **************/
/*********************************************************/
printf("\n\n*****************************************************************\n\n");

printf("Running simulation with the following Configuration:  \n\n");
printf("Network size (N)                              :     %d", NUM_ROUTERS_X );
printf("\nTotal Number of Routers (N*N)                :     %d",
                                              NUM_ROUTERS_X*NUM_ROUTERS_X );

printf("\nNumber of Processors                         :     %d",
                                              number_of_processors );

printf("\nSimulation Duration                          :     %d",
                                              SIMULATION_DURATION );

printf("\nPercent of packet injecting applications     :     %d", probability_i );
printf("\nDo routers absorb sleeping packets?          :     ");

if(absorb_sleeping_packet) {
printf("Yes");
} else {
printf("No");
}

printf("\nNumber of KPs                                :     %d", NUM_KP_Y );

printf("\n\n*****************************************************************\n\n");
/*********************************************************/
```

```
/*********************************************************/

g_tw_events_per_pe = (20 * number_of_lps / g_tw_npe) + 8192;
/* Num of Packets per node allowed * lps / Num of PEs + 4096 */

/*********************************************************/
/*                  LP lp mapping.                       */
/*********************************************************/
number_of_kps = (NUM_KP_X * NUM_KP_Y);

num_lps_per_kp = number_of_lps / number_of_kps;

/* Virtual Processes are basically kps per pe */

kp_per_proc = (int) (((double)NUM_KP_X * (double)NUM_KP_Y) /
                              (double)number_of_processors);

/*********************************************************/
/*  tw_init(tw_lptype, NumPE, NumKP, NumLP, Message_Size);
tw_lptype = the type of logical process (router in this sim)
NumPE = the number of processors in the computer
NumKP = the number of groups of lps
NumLP = the number of LPs
Message_Size = The amount of memory needed to store a message.  */
/*********************************************************/
if(tw_init(Router_lps, number_of_processors, number_of_kps, number_of_lps,
                                          sizeof(Msg_Data)) == 0)
{
      printf("Uh-oh\n");     /* Something went wrong. */
} else {

      for (x = 0; x < NUM_ROUTERS_X; x++)
      {
            for (y = 0; y < NUM_ROUTERS_Y; y++)
            {

                  for (i = 0; i < 4; i++)

      /*  We typecast kpx and ROUTERS x to doubles so we get a fraction  */
                  xkp = (int)((double)x * ((double)NUM_KP_X /
                        (double)NUM_ROUTERS_X));
                  ykp = (int)((double)y * ((double)NUM_KP_Y /
                        (double)NUM_ROUTERS_Y));

                  lp = tw_getlp(x + (y * NUM_ROUTERS_X));
                  kp = tw_getkp((x + (y * NUM_ROUTERS_X))/num_lps_per_kp);
                  tw_lp_settype(lp, Router);
                  tw_lp_onkp(lp, kp);
                  tw_lp_onpe(lp, tw_getpe((xkp + (ykp * NUM_KP_X)) / kp_per_proc));
                  tw_kp_onpe(kp, tw_getpe((xkp + (ykp * NUM_KP_X)) / kp_per_proc));

            }
      }
}
      #ifdef TRACER_CODE_ON
      printf("Check 1.\n");     /* Debug Code. */
      #endif
```

```
	tw_run(); /* Start Sim */


	/*****************************************************/
	/*  Print Statistics                                 */
	/*  Note:  We divide out the times by the num of lps.    */
	/*****************************************************/
	printf("\n*************************************************************\n\n");
	printf("Number of Routers                    :    %d\n\n", NumLps);

	printf("Number of packets delivered          :    %d\n", global_TotalDelivered);
	printf("Mean distance traversed              :    %f\n\n",
                        Mean_total_distance_traversed / NumLps);

	printf("Mean delivery time in time steps     :    %f\n", Mean_total_delivery_time
                        / NumLps /100);

	printf("Number of packets injected           :    %d\n",
                        global_total_packets_injected);

	printf("Number of time steps waited to inject :    %d\n",
                        global_total_injection_wait_time_steps);

	if(global_total_packets_injected) {

	printf("Average wait to inject a packet      :    %f\n", (double)
global_total_injection_wait_time_steps/(double) global_total_packets_injected);

	} else {
	printf("Average wait to inject a packet      :    %f\n", (double) 0);
	}
	printf("Worst Case wait to inject a packet   :    %d\n",
                        global_worst_case_injection_wait_time_steps);

	printf("\n*************************************************************\n\n");

	return 0;
}

/*************************************************************/
/*************************************************************/
/*************                                    ************/
/********             Router_StartUp Function           *********/
/*************                                    ************/
/*************************************************************/
/*************************************************************/
void  Router_StartUp(Router_State *SV, tw_lp * lp) {

	int i, row, column;
	tw_event *CurEvent;
	tw_stime ts;

	Msg_Data *NewM;
	SV->total_delivery_time = 0;
	SV->total_distance_traversed = 0;
	SV->total_packets_injected = 0;
```

```
SV->TotalDelivered = 0;
SV->worst_case_injection_wait_time_steps = 0;
SV->injection_wait_time_steps = 0;
SV->total_injection_wait_time_steps = 0;

/* SET LINKS TO FREE */
SV->link[NORTH] = FREE;
SV->link[SOUTH] = FREE;
SV->link[EAST]  = FREE;
SV->link[WEST]  = FREE;

      #ifdef TRACER_CODE_ON
                        printf("Starting LP : %d\n", lp->id);
      #endif


/*  Send a heartbeat event  */
#ifdef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
ts = TIME_STEP;
CurEvent = tw_event_new(lp, ts, lp);
NewM = (Msg_Data *) tw_event_data(CurEvent);
NewM->event_type = HEARTBEAT;
tw_event_send(CurEvent);
#endif

/* End heartbeat event */
if(!one_shot) {

      #ifdef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
      for(i = 0; i < 4; i++) {
      #endif

            if(   tw_rand_integer(lp->id, 0, (100000)*100) <=
                                    (100000)*probability_i ) {

                  /*  Send a PACKET_INJECTION_APPLICATION event  */
                  ts = TIME_STEP + INJECTING;
                  CurEvent = tw_event_new(lp, ts, lp);
                  NewM = (Msg_Data *) tw_event_data(CurEvent);
                  NewM->event_type = PACKET_INJECTION_APPLICATION;
                  tw_event_send(CurEvent);
                  /* End PACKET_INJECTION_APPLICATION event */
            }
      #ifdef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
      }
      #endif

}

for(i = 0; i < 4; i++) {

      ts = TIME_STEP+( (double) tw_rand_integer(lp->id, 1, 50000000)/100000000);
      CurEvent = tw_event_new(lp, ts, lp);
      NewM = (Msg_Data *) tw_event_data(CurEvent);
      NewM->event_type = ARRIVE;

      NewM->priority = SLEEPING;
```

```
            NewM->destination_row = tw_rand_integer(lp->id, 0, NumLpsRT-1);
            /* int rand 0 - NumLpsRT (inclusive) */
            NewM->destination_column = tw_rand_integer(lp->id, 0, NumLpsRT-1);
            /* int rand 0 - NumLpsRT (inclusive) */
            NewM->leave_time = tw_now(lp);
            column = lp->id % NumLpsRT;
            row = lp->id / NumLpsRT;

            NewM->distance =
                  min( max(column,NewM->destination_column)
                                    - min(column,NewM->destination_column),
                   (NUM_ROUTERS_X - max(column,NewM->destination_column))
                                    + min(column,NewM->destination_column) )
                  +
                  min( max(row,NewM->destination_row) - min(row,NewM->destination_row),
                   (NUM_ROUTERS_Y - max(row,NewM->destination_row)) + min(row,
                                    NewM->destination_row) );
            /* Debug Code
            printf("my row : %d\n", row);
            printf("my column : %d\n", column);
            printf("destination row : %d\n", NewM->destination_row);
            printf("destination column : %d\n", NewM->destination_column);
            printf("distance : %d\n", NewM->distance);
            */
            tw_event_send(CurEvent);

      }
}
/*************************************************************/
/*                      End of Startup                       */
/*************************************************************/

/*************************************************************/
/*******                 event    ARRIVE                 ********/
/*************************************************************/
/*
The ARRIVE event simulates the arrival of a packet.
The priority level is determined which dictates the order
in which the packets route will be considered by the router.
The LP then generates a appropriate message to itself
(destined for the same LP) which will initiate a route event.
If the packet arrives at its destination router, no new event
is created.  Instead, statistics regarding the event, such as
its delivery time, are recorded.
*/
void Router_ARRIVE_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {

      /* Declarations */
      tw_stime ts;
      tw_event *CurEvent;
      Msg_Data *NewM;
      int column, row;
      enum directions NewDir;
      NewDir = NO_DIRECTION;
      /*******************/

            #ifdef TRACER_CODE_ON
```

```
        printf("Check 8 (forward-arrive) at time: %f\n", tw_now(lp));
        /* Debug Code. */
        #endif

        #ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
        /* SET LINKS TO FREE */
              if( (CV->c5 = (SV->link[NORTH] == NOT_FREE)) ) {SV->link[NORTH]
                                = FREE;}
              if( (CV->c2 = (SV->link[SOUTH] == NOT_FREE)) ) {SV->link[SOUTH]
                                = FREE;}
              if( (CV->c3 = (SV->link[EAST]  == NOT_FREE)) ) {SV->link[EAST]
                                = FREE;}
              if( (CV->c4 = (SV->link[WEST]  == NOT_FREE)) ) {SV->link[WEST]
                                = FREE;}
        #endif
        /* Check Destination and Current Link */
        /* Figure out destination lp. */
        column = lp->id % NumLpsRT;
        row = lp->id / NumLpsRT;

        if( absorb_sleeping_packet || (!(M->priority == SLEEPING)) ) {
              if( (CV->c1 = ((M->destination_column == column)
                                && (M->destination_row == row))) ) {
                    /* collect statistics */
                    M->Saved_total_delivery_time =  SV->total_delivery_time;
                    /* double, must save */

                    SV->total_delivery_time += (tw_now(lp) - M->leave_time);

                    SV->total_distance_traversed += M->distance;
                    ++(SV->TotalDelivered);
                    /* eat packet */
                    return;/*break;*/
              }
        }

        /****     Generat new event     ****/
        ts = (M->priority + 1);
        CurEvent = tw_event_new(lp, ts, lp);

        /****     Set Up and Send Out Event     ****/
        NewM = (Msg_Data *)tw_event_data(CurEvent);
        NewM->event_type = ROUTE;
        NewM->priority     = M->priority;
        NewM->destination_row = M->destination_row;
        NewM->destination_column = M->destination_column;
        NewM->leave_time = M->leave_time;
        NewM->distance = M->distance;
        tw_event_send(CurEvent);
        /*****************************************************************/
        /*******       END       event    ARRIVE                 ********/
        /*****************************************************************/
}
/*********************************/
/***   Begin Case : ARRIVE RC   ***/
/*********************************/
void RC_Router_ARRIVE_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {
```

```
            #ifdef TRACER_CODE_ON
                        printf("Check 4 (RC-arrive) at time: %f\n", tw_now(lp));
                        /* Debug Code. */
            #endif

            #ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
            /* UN-SET LINKS TO FREE */
            if(CV->c5) {SV->link[NORTH] = NOT_FREE;}
            if(CV->c2) {SV->link[SOUTH] = NOT_FREE;}
            if(CV->c3) {SV->link[EAST]  = NOT_FREE;}
            if(CV->c4) {SV->link[WEST]  = NOT_FREE;}
            #endif

            if( (CV->c1) ) {
                  /* collect statistics */
            --(SV->TotalDelivered);
            SV->total_distance_traversed -= M->distance;
            SV->total_delivery_time = M->Saved_total_delivery_time;
            }
}           /*********************************/
            /***    End Case : ARRIVE RC    ***/
            /*********************************/

/*****************************************************************/
/*******            Get Any Free Link Function           ********/
/*****************************************************************/
/*    This function choses an arbitrary link
      if the preferred link is unavailable    */
enum directions Get_Any_Free_Link(Router_State *SV, enum bool traverse_row ) {

      enum directions NewDir;
      NewDir = NO_DIRECTION;

      if(traverse_row) {
                  /* Traverse Row preferred */


                  if (SV->link[EAST] == FREE) {
                              NewDir = EAST;
                        } else {

                              if (SV->link[NORTH] == FREE) {
                                    NewDir = NORTH;
                              } else {

                                    if (SV->link[SOUTH] == FREE) {
                                          NewDir = SOUTH;
                                    } else {

                                          if (SV->link[WEST] == FREE) {
                                                NewDir = WEST;
                                          } else {
                                                printf("Sim. Failed Check -
                                                More than 4 packets at router.\n");
                                          }
```

```
                                    }
                              }

                        }

            } else {  /* traverse column */

                  if (SV->link[NORTH] == FREE) {
                        NewDir = NORTH;

                  } else {

                        if (SV->link[EAST] == FREE) {
                              NewDir = EAST;
                        } else {

                              if (SV->link[WEST] == FREE) {
                                    NewDir = WEST;
                              } else {

                                          if (SV->link[SOUTH] == FREE) {
                                                NewDir = SOUTH;
                                          } else {
                                                printf("Sim. Failed Check
                                                - More than 4 packets at router.\n");
                                          }
                              }
                        }
                  }
            }
            return NewDir;
}


/*****************************************************************/
/*******        Traverse_Good_Links_Row_Preferred         ********/
/*****************************************************************/
/*    This function choses one of the "good" links for the packet.
      It "prefers" to traverse the row first.
        But, it will traverse the column if necessary.  */
inline enum bool Traverse_Good_Links_Row_Preferred(Router_State *SV, tw_bf *CV, Msg_Data
*M, tw_lp *lp, int row, int column, enum directions *NewDir ) {


      if(M->destination_column != column) {
                  /* Traverse Row_PREFERRED */


                  if( ((M->destination_column + NUM_ROUTERS_X - column) % NUM_ROUTERS_X)
<
                        ((column - M->destination_column + NUM_ROUTERS_X)
                                    % NUM_ROUTERS_X) ) {
                        /*    try east; */

                        if (SV->link[EAST] == FREE) {
                              *NewDir = EAST;
                              return true;
```

```
                        }

                  }

                  if( ((M->destination_column + NUM_ROUTERS_X - column)
                        % NUM_ROUTERS_X) >=
                      ((column - M->destination_column + NUM_ROUTERS_X)
                        % NUM_ROUTERS_X) ) {
                        /*    try west; */
                        if (SV->link[WEST] == FREE) {
                              *NewDir = WEST;
                        return true;
                        }

                  }
      }
      if(M->destination_row != row) {

      if( ((M->destination_row + NUM_ROUTERS_Y - row) % NUM_ROUTERS_Y) <
            ((row - M->destination_row + NUM_ROUTERS_Y) % NUM_ROUTERS_Y) ) {
                  /*    try south; */
                  if (SV->link[SOUTH] == FREE) {
                        *NewDir = SOUTH;
                              return true;
                  }

            }


            if( ((M->destination_row + NUM_ROUTERS_Y - row) % NUM_ROUTERS_Y) >=
                  ((row - M->destination_row + NUM_ROUTERS_Y) % NUM_ROUTERS_Y) ) {
                        /*    try north; */
                        if (SV->link[NORTH] == FREE) {
                              *NewDir = NORTH;
                              return true;
                        }
                  }
      }

      return false;

}/*************************************************************/
/*******   END  Traverse_Good_Links_Row_Preferred         ********/
/*************************************************************/

/*************************************************************/
/*******        Traverse_Good_Links_Row_First             ********/
/*************************************************************/
/*    This function choses one of the "good" links for the packet.
      It insists on traversing the row first.
        It will not traverse the column until it is in its home-run (destination) column.
        If it cannot traverse the row, it is considered "interupted".  */
inline enum bool Traverse_Good_Links_Row_First(Router_State *SV, tw_bf *CV, Msg_Data *M,
tw_lp *lp, int row, int column, enum directions *NewDir ) {

      if(M->destination_column != column) {
            /* Traverse Row_FIRST */
```

```
                if( ((M->destination_column + NUM_ROUTERS_X - column) % NUM_ROUTERS_X)

                      <

                      ((column - M->destination_column + NUM_ROUTERS_X)
                                  % NUM_ROUTERS_X) ) {
                            /*    try east; */

                            if (SV->link[EAST] == FREE) {
                                  *NewDir = EAST;
                                  return true;
                            }
                      }

                if( ((M->destination_column + NUM_ROUTERS_X - column) % NUM_ROUTERS_X)

                      >=

                ((column - M->destination_column + NUM_ROUTERS_X) % NUM_ROUTERS_X) ) {
                            /*    try west; */
                            if (SV->link[WEST] == FREE) {
                                  *NewDir = WEST;
                                  return true;
                            }
                      }
          } else { /* only if destination column == column */

          if( ((M->destination_row + NUM_ROUTERS_Y - row) % NUM_ROUTERS_Y) <=
                ((row - M->destination_row + NUM_ROUTERS_Y) % NUM_ROUTERS_Y) ) {
                      /*    try south; */
                      if (SV->link[SOUTH] == FREE) {
                            *NewDir = SOUTH;
                            return true;
                      }
                }

          if( ((M->destination_row + NUM_ROUTERS_Y - row) % NUM_ROUTERS_Y) >=
                ((row - M->destination_row + NUM_ROUTERS_Y) % NUM_ROUTERS_Y) ) {
                      /*    try north; */
                      if (SV->link[NORTH] == FREE) {
                            *NewDir = NORTH;
                            return true;
                      }
                }
      }

      return false;

}/*************************************************************/
/*******   END  Traverse_Good_Links_Row_First         ********/
/*************************************************************/


/*************************************************************/
/*******                  event    ROUTE                    ********/
/*************************************************************/
```

```
/*
The ROUTE event determines which direction the packet will be routed.
It also determines if the packets priority will be changed,
as described in the algorithm description above.
It then creates a new arrive event at the appropriate
destination router.
*/
void Router_ROUTE_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {
            #ifdef TRACER_CODE_ON
            printf("Check 9 (forward-route) at time: %f\n", tw_now(lp));
            /* Debug Code. */
            #endif
      /* Declarations */
      tw_stime ts;
      tw_event *CurEvent;
      Msg_Data *NewM;
      int NewLp, column, row;
      enum directions NewDir;
      enum bool deflected;
      NewDir = NO_DIRECTION;
      NewLp = 0;
      /********************/
      deflected = false;

            ts = (TIME_STEP - (M->priority + 1) );
            /* Figure out lp row and column. */
            column = lp->id % NumLpsRT;
            row = lp->id / NumLpsRT;

            /* Choose new direction. */
            switch(M->priority) {

      case SLEEPING:

            /* TRY GOOD LINKS */
            /* Traverse Row_PREFERRED */
            if ( Traverse_Good_Links_Row_Preferred( SV, CV, M, lp, row, column, &NewDir)
) {
                  deflected = false;
                  break;
            }
            /* TRIED ALL GOOD LINKS. */
            /* Traverse and free link. */
            NewDir = Get_Any_Free_Link(SV,M->destination_column != column);

            deflected = true;

            break;

      case ACTIVE:

            /* TRY GOOD LINKS */
            /* Traverse Row_PREFERRED */
            if ( Traverse_Good_Links_Row_Preferred( SV, CV, M, lp, row, column, &NewDir)
) {
                  deflected = false;
                  break;
```

```
        }
        /* TRIED ALL GOOD LINKS. */
        /* Traverse and free link. */
              NewDir = Get_Any_Free_Link(SV,M->destination_column != column);

              deflected = true;

        break;

case EXCITED:

        /* TRY GOOD LINKS */
        /* Traverse Row_First */
        if ( Traverse_Good_Links_Row_First( SV, CV, M, lp, row, column, &NewDir) ) {

              deflected = false;
              break;
        }

        /* TRIED ALL GOOD LINKS. */
        /* Traverse and free link. */
              NewDir = Get_Any_Free_Link(SV,M->destination_column != column);
              deflected = true;

        break;

case RUNNING:

        /* TRY GOOD LINKS     */
        /* Traverse Row_First */
        if ( Traverse_Good_Links_Row_First( SV, CV, M, lp, row, column, &NewDir) ) {

              deflected = false;
              break;
        }
        /* TRIED ALL GOOD LINKS.   */
        /* Traverse and free link. */
                    /* TRIED ALL GOOD LINKS */

        NewDir = Get_Any_Free_Link(SV,M->destination_column != column);

        deflected = true;

        break;

case RUNNING_HOME:

        /* TRY GOOD LINKS     */
        /* Traverse Row_First */
        if ( Traverse_Good_Links_Row_First( SV, CV, M, lp, row, column, &NewDir) ) {

              deflected = false;
              break;
        }
        /* TRIED ALL GOOD LINKS.   */
        /* Traverse and free link. */
                    /* TRIED ALL GOOD LINKS */
```

```
      NewDir = Get_Any_Free_Link(SV,M->destination_column != column);

      deflected = true;

      break;

case TURNING:
      break;

case PLACE_HOLDER:
      break;

case INJECTING:


      /* TRY GOOD LINKS         */
      /* Traverse Row_PREFERRED */
   if ( Traverse_Good_Links_Row_Preferred( SV, CV, M, lp, row, column, &NewDir) ) {

            deflected = false;
            break;
      }
      /* TRIED ALL GOOD LINKS.   */
      /* Traverse and free link. */
      NewDir = Get_Any_Free_Link(SV,M->destination_column != column);

            deflected = true;

            break;

      } /* END PICK NEW DIR (PRIORITY) SWITCH */


      if( (CV->c3 = (NewDir == NORTH)) ) {
            SV->link[NORTH] = NOT_FREE;
      }
      if( (CV->c4 = (NewDir == SOUTH)) ) {
            SV->link[SOUTH] = NOT_FREE;
      }
      if( (CV->c5 = (NewDir == EAST)) ) {
            SV->link[EAST] = NOT_FREE;
      }
      if( (CV->c6 = (NewDir == WEST)) ) {
            SV->link[WEST] = NOT_FREE;
      }


      /* Compute lp->id. */
      switch(NewDir) {

case NORTH:
      NewLp = (NumLps + (lp->id - NumLpsRT)) % NumLps;
      break;
case SOUTH:
      NewLp = (lp->id + NumLpsRT) % NumLps;
      break;
```

```
case EAST:
      NewLp = ((lp->id / NumLpsRT) * NumLpsRT) + ((lp->id + 1) % NumLpsRT);
      break;
case WEST:
NewLp = ((lp->id / NumLpsRT) * NumLpsRT) + ((NumLpsRT + (lp->id - 1)) % NumLpsRT);
      break;
case NO_DIRECTION:
      printf("Error: Packet with NO_DIRECTION detected.\n");
      break;

      }
      if(NewDir == NO_DIRECTION){
            printf("Error: Break without sending packet.\n");
            return; /*break;*/
      }


      CurEvent = tw_event_new(tw_getlp(NewLp), ts, lp);

      NewM = (Msg_Data *)tw_event_data(CurEvent);
      NewM->event_type = ARRIVE;
      /* set priority */

      NewM->priority     = M->priority;

      if( (CV->c1 = (M->priority == SLEEPING)) ) {
      /* possibly set to active */
            /*1/24n*/

            if( (tw_rand_integer(lp->id, 0, (NUM_ROUTERS_X-1)*probability_q)
                  <= (NUM_ROUTERS_X-1)) ) {

                  NewM->priority = ACTIVE;
                  /*printf("reset from sleeping to active\n");*/
            }

      }

      if( (CV->c2 =  ((M->priority == ACTIVE) && deflected)  ) ) {
      /* possibly set to active */
            /*1/24n*/

            if(   (tw_rand_integer(lp->id, 0, (NUM_ROUTERS_X-1)*probability_p)
                  <= (NUM_ROUTERS_X-1)) ) {

                  NewM->priority = EXCITED;
            /*    printf("reset from ACTIVE to EXCITED\n");  */
            }
      }

      if( (M->priority == EXCITED ) && deflected ) {
                  NewM->priority = ACTIVE;
            /*    printf("reset from EXCITED to ACTIVE\n");  */
      }
      if( (M->priority == EXCITED ) && !deflected ) {
                  NewM->priority = RUNNING;
            /*    printf("reset from EXCITED to RUNNING\n");  */
```

```
		}

		if( (column == M->destination_column) && (M->priority == RUNNING )
				&& !deflected ) {

				NewM->priority = RUNNING_HOME;
			/*	printf("reset from RUNNING to RUNNING_HOME.\n");  */
		}

		if( (M->priority == RUNNING_HOME ) && deflected ) {
				NewM->priority = ACTIVE;
			/*	printf("reset from RUNNING_HOME to ACTIVE while turning.\n");  */
		}

		if( (M->priority == RUNNING ) && deflected ) {
				NewM->priority = ACTIVE;
			/*	printf("reset from RUNNING to ACTIVE while turning.\n");  */
		}

		if( (M->priority == INJECTING ) ) {
				NewM->priority = SLEEPING;
/*				printf("reset from INJECTING to SLEEPING.\n");
*/		}

		NewM->destination_row = M->destination_row;
		NewM->destination_column = M->destination_column;
		NewM->leave_time = M->leave_time;
		NewM->distance = M->distance;
		tw_event_send(CurEvent);

		#ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
		if(M->priority == INJECTING) {
		/* SET LINKS TO FREE */
		if( (CV->c7  = (SV->link[NORTH] == NOT_FREE)) ) {SV->link[NORTH] = FREE;}
		if( (CV->c8  = (SV->link[SOUTH] == NOT_FREE)) ) {SV->link[SOUTH] = FREE;}
		if( (CV->c9  = (SV->link[EAST]  == NOT_FREE)) ) {SV->link[EAST]  = FREE;}
		if( (CV->c10 = (SV->link[WEST]  == NOT_FREE)) ) {SV->link[WEST]  = FREE;}
		}
		#endif

/*************************************************************/
/*******         END     event    ROUTE                 ********/
/*************************************************************/
}

/*******************************/
/***   Begin Case : ROUTE RC  ***/
/*******************************/
void RC_Router_ROUTE_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {

		#ifdef TRACER_CODE_ON
		printf("Check 5 (RC-route) at time: %f\n", tw_now(lp));     /* Debug Code. */
		#endif

		#ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
		if(M->priority == INJECTING) {
```

```
            /* UN-SET LINKS TO FREE */
            if(CV->c7)  {SV->link[NORTH] = NOT_FREE;}
            if(CV->c8)  {SV->link[SOUTH] = NOT_FREE;}
            if(CV->c9)  {SV->link[EAST]  = NOT_FREE;}
            if(CV->c10) {SV->link[WEST]  = NOT_FREE;}
            }
            #endif
            if( CV->c1 ) { tw_rand_reverse_unif(lp->id); }
            if( CV->c2 ) { tw_rand_reverse_unif(lp->id); }

            if( CV->c3 ) {
                  SV->link[NORTH] = FREE;
            }
            if( CV->c4 ) {
                  SV->link[SOUTH] = FREE;
            }
            if( CV->c5 ) {
                  SV->link[EAST] = FREE;
            }
            if( CV->c6 ) {
                  SV->link[WEST] = FREE;
            }

            /*******************************/
            /***    End Case : ROUTE RC   ***/
            /*******************************/
}

/*************************************************************/
/*******                  event    HEARTBEAT              ********/
/*************************************************************/
/*
The HEARTBEAT event simply generates events to perform
administrative overhead.  In some configurations, that
overhead can be taken care of by other events.
In those cases, the HEARTBEAT event is not used,
so as to reduce the total number of simulated events.
*/
void Router_HEARTBEAT_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {

            #ifdef TRACER_CODE_ON
                  printf("Check 10 (forward-heartbeat) at time: %f\n", tw_now(lp));
/* Debug Code. */
            #endif

      /* Declarations */
      tw_stime ts;
      tw_event *CurEvent;
      Msg_Data *NewM;
      enum directions NewDir;
      NewDir = NO_DIRECTION;
      /********************/

            /* SET LINKS TO FREE */
                  if( (CV->c1 = (SV->link[NORTH] == NOT_FREE)) ) {
                        SV->link[NORTH] = FREE;}
                  if( (CV->c2 = (SV->link[SOUTH] == NOT_FREE)) ) {
```

```
                        SV->link[SOUTH] = FREE;}
                  if( (CV->c3 = (SV->link[EAST]  == NOT_FREE)) ) {
                        SV->link[EAST]  = FREE;}
                  if( (CV->c4 = (SV->link[WEST]  == NOT_FREE)) ) {
                        SV->link[WEST]  = FREE;}

            /*  Send a heartbeat event  */
            ts = TIME_STEP;
            CurEvent = tw_event_new(lp, ts, lp);
            NewM = (Msg_Data *) tw_event_data(CurEvent);
            NewM->event_type = HEARTBEAT;
            tw_event_send(CurEvent);
            /* End heartbeat event */
}           /*****************************************************************/
            /*******       END       event    HEARTBEAT               ********/
            /*****************************************************************/


/***********************************/
/***   Begin Case : HEARTBEAT RC   ***/
/***********************************/
void RC_Router_HEARTBEAT_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M,
            tw_lp *lp) {

            #ifdef TRACER_CODE_ON
            printf("Check 6 (RC-heartbeat) at time: %f\n", tw_now(lp));
            /* Debug Code. */
            #endif

            /* UN-SET LINKS TO FREE */
            if(CV->c1) {SV->link[NORTH] = NOT_FREE;}
            if(CV->c2) {SV->link[SOUTH] = NOT_FREE;}
            if(CV->c3) {SV->link[EAST]  = NOT_FREE;}
            if(CV->c4) {SV->link[WEST]  = NOT_FREE;}
            /* No OTHER state variables to RC. */
            /***********************************/
            /***    End Case : HEARTBEAT RC    ***/
            /***********************************/

}

/*************************************************************/
/*******      event   PACKET_INJECTION_APPLICATION       ********/
/*************************************************************/
/*
The PACKET_INJECTION_APPLICATION event simulates the
injection of new packets into the system.  The number
of LPs that are “packet generators” is determined by the
startup program based on the application input parameters.
Anywhere fro zero to n*n events can be packet generators.
In our tests, n events are packet generators.
This simulates a scenario where the network is kept
relatively full, yet there are still specific sources.
*/
```

```
void Router_PACKET_INJECTION_APPLICATION_EventHandler(Router_State *SV,
                                                      tw_bf *CV,
                                                      Msg_Data *M,
                                                      tw_lp *lp) {

            #ifdef TRACER_CODE_ON
            printf("Check 11 (forward-injection) at time: %f\n", tw_now(lp));
            /* Debug Code. */
            #endif

      /* Declarations */
      tw_stime ts;
      tw_event *CurEvent;
      Msg_Data *NewM;
      int column, row;
      enum directions NewDir;
      NewDir = NO_DIRECTION;
      /********************/

                  /* if there is a free link, then inject your packet */
                  if ( (CV->c1 = (    (SV->link[NORTH] == FREE)
                                       || (SV->link[SOUTH] == FREE)
                                       || (SV->link[EAST]  == FREE)
                                       || (SV->link[WEST]  == FREE)) ) ) {

                        /*  record delay time */
                        SV->total_injection_wait_time_steps
                        += SV->injection_wait_time_steps;

                        M->Saved_worst_case_injection_wait_time_steps
                              = SV->worst_case_injection_wait_time_steps;

                        SV->worst_case_injection_wait_time_steps
                              = max(SV->worst_case_injection_wait_time_steps,
                                    SV->injection_wait_time_steps);

                        M->Saved_injection_wait_time_steps
                              = SV->injection_wait_time_steps;

                        SV->injection_wait_time_steps = 0;
                        ++(SV->total_packets_injected);

            /*  inject PACKET  */
            /* Add random 1/2 ts */
            ts = 1+ ( (double) tw_rand_integer( lp->id, 1, 50000000)/100000000);
            CurEvent = tw_event_new(lp, ts, lp);
            NewM = (Msg_Data *) tw_event_data(CurEvent);
            NewM->event_type = ROUTE;

            NewM->priority = INJECTING;
            /* int rand 0 - NumLpsRT (inclusive) */
            NewM->destination_row = tw_rand_integer(lp->id, 0, NumLpsRT-1);
            /* int rand 0 - NumLpsRT (inclusive) */
            NewM->destination_column = tw_rand_integer(lp->id, 0, NumLpsRT-1);
            NewM->leave_time = tw_now(lp) - INJECTING;
            column = lp->id % NumLpsRT;
            row = lp->id / NumLpsRT;
```

```
            /*  Calculate the distance from the injecting node to
                  the destination node.  */
            NewM->distance =
                  min( max(column,NewM->destination_column)
                  - min(column,NewM->destination_column),
                  (NUM_ROUTERS_X - max(column,NewM->destination_column))
                  + min(column,NewM->destination_column) )
            +
                  min( max(row,NewM->destination_row)
                  - min(row,NewM->destination_row),
                  (NUM_ROUTERS_Y - max(row,NewM->destination_row))
                  + min(row, NewM->destination_row) );

            tw_event_send(CurEvent);
                  /*  done - SEND PACKET  */

                  } else { /* If we can't inject now, then: */
                        /*  INCREMENT INJECTION DELAY TIME */
                        ++(SV->injection_wait_time_steps);

                        #ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
                        /* SET LINKS TO FREE */
                        if( (CV->c5 = (SV->link[NORTH] == NOT_FREE)) ) {
                              SV->link[NORTH] = FREE;}

                        if( (CV->c2 = (SV->link[SOUTH] == NOT_FREE)) ) {
                              SV->link[SOUTH] = FREE;}

                        if( (CV->c3 = (SV->link[EAST]  == NOT_FREE)) ) {
                              SV->link[EAST]  = FREE;}

                        if( (CV->c4 = (SV->link[WEST]  == NOT_FREE)) ) {
                              SV->link[WEST]  = FREE;}
                        #endif

                  }

      /*  Send a PACKET_INJECTION_APPLICATION event  */
            ts = TIME_STEP;
            CurEvent = tw_event_new(lp, ts, lp);
            NewM = (Msg_Data *) tw_event_data(CurEvent);
            NewM->event_type = PACKET_INJECTION_APPLICATION;
            tw_event_send(CurEvent);
      /* End PACKET_INJECTION_APPLICATION event */


}

/*****************************************************/
/***   Begin Case : PACKET_INJECTION_APPLICATION RC   ***/
/*****************************************************/
void RC_Router_PACKET_INJECTION_APPLICATION_EventHandler(Router_State *SV,

tw_bf *CV,

Msg_Data *M,
```

```
tw_lp *lp) {

            #ifdef TRACER_CODE_ON
            printf("Check 7 (RC-injection) at time: %f\n", tw_now(lp));     /* Debug
Code. */
            #endif

                  /* if there is a free link, then inject your packet */

                  if ( (CV->c1) ) {

                        tw_rand_reverse_unif(lp->id);
                        tw_rand_reverse_unif(lp->id);
                        tw_rand_reverse_unif(lp->id);

                        --(SV->total_packets_injected);
                        SV->injection_wait_time_steps
                              = M->Saved_injection_wait_time_steps;

                        SV->worst_case_injection_wait_time_steps
                              = M->Saved_worst_case_injection_wait_time_steps;

                        SV->total_injection_wait_time_steps
                              -= SV->injection_wait_time_steps;

                  } else {

                        --(SV->injection_wait_time_steps);

                        #ifndef ROUTERS_CAN_INJECT_MULTIPLE_PACKETS
                      /* UN-SET LINKS TO FREE */
                        if(CV->c5)  {SV->link[NORTH] = NOT_FREE;}
                        if(CV->c2)  {SV->link[SOUTH] = NOT_FREE;}
                        if(CV->c3)  {SV->link[EAST]  = NOT_FREE;}
                        if(CV->c4)  {SV->link[WEST]  = NOT_FREE;}
                        #endif

                  }
}           /*********************************************************/
            /***    End Case : PACKET_INJECTION_APPLICATION RC    ***/
            /*********************************************************/



/*************************************************************/
/*************************************************************/
/*************                                  **************/
/********          Router_EventHandler Function          *********/
/*************                                  **************/
/*************************************************************/
/*************************************************************/
/*
This function handles all events.
It calls the appropriate function depending on
the event type.
*/
```

```
void Router_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {

	enum directions NewDir;
	enum bool deflected;
	NewDir = NO_DIRECTION;

	/* reset bit fields CV->* to 0 for this event */
	*(int *)CV = (int)0;

	deflected = false;

	switch(M->event_type) {

	case ARRIVE:

		Router_ARRIVE_EventHandler( SV, CV, M, lp );

		break;

	case ROUTE:

		Router_ROUTE_EventHandler( SV, CV, M, lp );

		break;


	case HEARTBEAT:

		Router_HEARTBEAT_EventHandler( SV, CV, M, lp );

		break;


	case PACKET_INJECTION_APPLICATION:

		Router_PACKET_INJECTION_APPLICATION_EventHandler( SV, CV, M, lp );

		break;

	}
}

/*************************************************************/
/*************************************************************/
/*********                                          **********/
/*********       Router_RC_EventHandler Function        *********/
/*********                                          **********/
/*************************************************************/
/*************************************************************/
void Router_RC_EventHandler(Router_State *SV, tw_bf *CV, Msg_Data *M, tw_lp *lp) {

	#ifdef TRACER_CODE_ON
	printf("Check 2 (RC) at time: %f\n", tw_now(lp));	/* Debug Code. */
	#endif


	switch(M->event_type) {
```

```
	case ARRIVE:
		RC_Router_ARRIVE_EventHandler( SV, CV, M, lp );
		break;
	case ROUTE:
		RC_Router_ROUTE_EventHandler( SV, CV, M, lp );
		break;

	case HEARTBEAT:
		RC_Router_HEARTBEAT_EventHandler( SV, CV, M, lp );
		break;

	case PACKET_INJECTION_APPLICATION:
		RC_Router_PACKET_INJECTION_APPLICATION_EventHandler( SV, CV, M, lp );
		break;
	}
}

/*****************************************************************/
/*****************************************************************/
/*********                                              **********/
/********     Router_Statistics_CollectStats Function    *********/
/*********                                              **********/
/*****************************************************************/
/*****************************************************************//*
 Calculate Statistics :

 This simulation collects several statistics.
 In particular, we want to know what the expected packet
 delivery time is with respect to the network size.
 Therefore, each router keeps track of the total number
 of packets that were delivered to it, how long the packets
 were in transit, and how far they came.

We also want to know how long a packet waits to be injected
into the network(expected and worst case time).
Therefore, each router keeps track of the amount
of time that each injected packets waited to be injected,
the total number of packets that were injected into the system,
and the longest time that any injected packed had to wait to be
injected at that router.
*/

void Router_Statistics_CollectStats(Router_State *SV, tw_lp * lp) {
	#ifdef TRACER_CODE_ON
```

```
            printf("Check 3 (collect stats) at time: %f\n", tw_now(lp));
            /* Debug Code. */
            printf("SV->total_delivery_time / SV->TotalDelivered: %f\n",
                  SV->total_delivery_time / SV->TotalDelivered);
            printf("SV->total_distance_traversed / SV->TotalDelivered : %f\n",
                  SV->total_distance_traversed / SV->TotalDelivered);
            printf("SV->total_distance_traversed : %f\n", SV->total_distance_traversed );
            printf("SV->TotalDelivered : %d\n", SV->TotalDelivered);
      #endif

            if(SV->TotalDelivered) {
            Mean_total_delivery_time += SV->total_delivery_time / SV->TotalDelivered;

      /*  Sum up the total distance traversed for all packets sent and recieved.  */

            Mean_total_distance_traversed += SV->total_distance_traversed /
                        SV->TotalDelivered;
            }

      global_TotalDelivered += SV->TotalDelivered;
      global_total_injection_wait_time_steps += SV->total_injection_wait_time_steps;
      global_total_packets_injected += SV->total_packets_injected;
      global_worst_case_injection_wait_time_steps
            = max(SV->worst_case_injection_wait_time_steps,
            global_worst_case_injection_wait_time_steps);


}
```