```
PROCEDURE Prune_dead_ends (VAR Network_defn : Networks;j : Node_ID);
{=========================================================}

  {*****************************************************}
  {*                                                   *}
  {* This procedure is to find the sink tree from a parti- *}
  {* source destination pair.                          *}
  {*                                                   *}
  {* INPUTS:  Variables: The matrix defining the sink tree *}
  {*          Constants: The network definition        *}
  {* OUTPUTS: Variables: The matrix defining the sink tree *}
  {*                                                   *}
  {*****************************************************}

  VAR k, a, neighbour_node : Node_ID;
      dead_end             : BOOLEAN;
      All_done             : BOOLEAN;

  BEGIN
    REPEAT

      All_done := TRUE;

      FOR a := 1 TO Network_defn.Number_of_nodes DO
        IF (a <> j)
        THEN
          BEGIN
            dead_end := TRUE;

            FOR neighbour_node := 1 TO Network_defn.Number_of_nodes DO
              IF ((neighbour_node <> a)
                AND (Network_defn.sink_tree[a,neighbour_node,j] = Output_path))
              THEN dead_end := FALSE;

            IF (dead_end = TRUE)
            THEN
              FOR k := 1 TO Network_defn.Number_of_nodes DO
                IF ((k <> a) AND (Network_defn.sink_tree[k,a,j] = Output_path))
                THEN
                  BEGIN
                    Network_defn.sink_tree[k,a,j] := Not_connected;
                    Network_defn.sink_tree[a,k,j] := Output_path;
                    All_done                      := FALSE;
                  END;

          END; {If a <> j}

      UNTIL (All_done = TRUE);
    END;
```

```
PROCEDURE Find_more_paths (VAR Network_defn : Networks);
{=========================================================}

  {*****************************************************}
  {*                                                   *}
  {* The function of this procedure is to  find additional *}
  {* loop free paths in the network.                   *}
  {*                                                   *}
  {* INPUTS:  Variable: The sink tree for the network. *}
  {* OUTPUTS: Variable: The updated sink tree.         *}
  {*                                                   *}
  {*****************************************************}

  VAR i, k, j          : Node_ID;
      No_of_OP_paths : Node_ID;
      add_paths      : BOOLEAN;

  BEGIN

    FOR i := 1 TO Network_defn.Number_of_nodes DO
      FOR j := 1 TO Network_defn.Number_of_nodes DO
        BEGIN

          { Check to see if there no incoming links }
          {-----------------------------------------}

          IF (i <> j)
          THEN
            BEGIN
              add_paths := TRUE;

              FOR k := 1 TO Network_defn.Number_of_nodes DO
                IF ((i <> k) AND (Network_defn.capacities[i,k] > 0)
                  AND (Network_defn.Sink_tree[k,i,j] = Output_path))
                THEN add_paths := FALSE;


              { If there are no incoming links, Check if more than one output is available }
              {-------------------------------------------------------------------------}

              IF (add_paths)
              THEN
                BEGIN
                  No_of_OP_paths := 0;

                  FOR k := 1 TO Network_defn.Number_of_nodes DO
                    IF (network_defn.Sink_tree[i,k,j] = Output_path)
                    THEN No_of_OP_paths := No_of_OP_paths + 1;

                  IF (No_of_OP_paths < 2) THEN add_paths := FALSE;
                END;
```

```
                    { If more than one O/P path and 0 I/P paths, then add I/P paths }
                    {-------------------------------------------------------------}
                      IF Load_paths
                      THEN
                        FOR k := 1 TO Network_defn.Number_of_nodes DO
                          IF (k <> j) AND (Network_defn.capacities[k,1] > 0) AND (k <> 1)
                          THEN
                            BEGIN
                              Network_defn.Sink_tree[k,1,1] := Output_path;
                              writeln('Path added from node ',k:3,' to node ',1:3);
                            END;

                  END; {If i <> j}
            END; {For j}
        END; {Find_more_paths}


    PROCEDURE Show_sink_tree (Network_defn : Networks; j : Node_ID);
    {=============================================================}

      {*********************************************************}
      {*                                                       *}
      {* This purpose of the procedure is to display the sink  *}
      {* tree for debugging purposes.                          *}
      {*                                                       *}
      {* INPUTS:  Variables: The matrix defining the sink tree *}
      {*          Constant: The network definition             *}
      {* OUTPUTS: Screen:   The matrix defining the sink tree  *}
      {*                                                       *}
      {*********************************************************}

      VAR k, a : NODE_ID;

      BEGIN

        FOR a := 1 TO Network_defn.Number_of_nodes DO
          BEGIN
            FOR k := 1 TO Network_defn.Number_of_nodes DO
              IF (Network_defn.Sink_tree[a,k,1] = Output_path)
              THEN write('OUT  ')
              ELSE IF (Network_defn.Sink_tree[a,k,1] = Not_connected)
                   THEN write('N/C  ')
                   ELSE write('XXX  ');
            writeln;
          END;
        writeln;
      END;
```

```
      IF (Tot_No_of_hops <> 0)
      THEN Total_hop_delay       := Tot_hop_delay/Tot_no_of_hops;

      Total_time                 := Total_service_time    + Total_dept_time + Total_Tx_time;
      Delay_per_pkt_gen          := Overall_message_delay + (Tot_msgs_Tx / Total_pkts_generated);
      Total_service_time         := (Total_service_time   * 100)/Total_time;
      Total_dept_time            := (Total_dept_time      * 100)/Total_time;
      Total_Tx_time              := (Total_Tx_time        * 100)/Total_time;
      Percentage_Pkts_blocked    := ((No_of_pkts_blocked  * 100)/Total_pkts_Tx);
      Total_messages             := Tot_msgs_Tx;
    END;


  { Calculate percentage use of each link }
  {-------------------------------------}
    FOR i := 1 TO No_of_intervals DO
      BEGIN
        Interval_duration := Stats.Interval_time[i+1] - Stats.Interval_time[i];
        No_of_links := 0; Total_use := 0;
        FOR x := 1 TO N DO
          FOR y := 1 TO N DO
            IF ((Network_defn.capacities[x,y] > 0) AND (Interval_duration > 0))
              THEN
                BEGIN
                  Stats.Percentage_usels,y,i] :=
                    ((Stats.Packets_per_link[x,y,i]  * Network_defn.bp_packet)/
                     (Interval_duration*Network_defn.capacities[x,y]*1000))*100;
                  Total_use := Total_use + Stats.Percentage_usels,y,i];
                  INC(No_of_links);
                END;
        IF (No_of_links > 0) THEN Stats.Average_percentage_use[i] := Total_use/No_of_links;
      END; { For i }

  END;



PROCEDURE Send_OP_to_file (N              : Node_ID;     Stats    : Stats_info;
                           No_of_periods  : Intervals;   Duration : Real;
                           ID_sentence    : ID_sentence; Result_file : filename);
{=============================================================================}

  {*************************************************}
  {*                                               *}
  {* This procedure sends the results of the simulation to *}
  {* an output file.                               *}
  {*                                               *}
  {*                                               *}
  {* INPUTS:  Variables: Stats data                *}
  {* OUTPUTS: Files                                *}
  {*                                               *}
  {*************************************************}
```

```
VAR     statusF             : ENTIER32;
        F                   : TEXT;
        filename            : ARRAY[1..30] OF CHAR;
        Max_no_of_delays    : INTEGER;
        x,y,i               : INTEGER;

BEGIN
  OPEN(F,Result_file,'UNKNOWN',statusF);
  IF (statusF = 0)
  THEN rewrite(F)
  ELSE
    BEGIN
      writeln('ERROR>> Cannot open output file'); writeln;
      RETURN;
    END;

  writeln(F,ID_name); writeln(F);
  writeln(F,'Simulation results');
  writeln(F,'==========');  writeln(F);
  writeln(F,'Algorithm ---> ',Stats.Selector_algorithm);
  IF (Stats.Selector_algorithm = New_algo)
  THEN
    BEGIN
      writeln(F,'Piggy Backing on input and output queues AND traffic prediction used');
      writeln(F,' Max a = ',Max_a:4,'      Alpha = ',Alpha:6:2);
    END;
  writeln(F);


  { Write out the general results for the entire simulation }
  {-------------------------------------------}
    writeln(F,'General Results');
    writeln(F,'---------------');
    write(F,'No.    Time     Delays  Pkts_blocked  Delay/PktTx  Cost/PktTx    Hop_delay');
    writeln(F,'    Msgs');

    FOR i := 1 TO No_of_periods DO
      IF (Stats.Interval_Time[i] > 0)
      THEN
        BEGIN
          write(F,i:3);
          write(F,Stats.Interval_Time[i]:10:4);
          write(F,Stats.Total_message_delays[i]:10:4);
          write(F,Stats.Total_packets_blocked[i]:8);
          write(F,Stats.Average_message_delay[i]:13:4);
          write(F,Stats.Total_cost[i]:14:4);
          write(F,Stats.Hop_delay[i]:14:4);
          write(F,Stats.Messages_transmitted[i]:8);
          writeln(F);
        END;
    writeln(F);
```

```
{ Write out the simulation results with respect to originating intervals }
{-------------------------------------------------------------------------}
  writeln(F,'Interval Results');
  writeln(F,'----------------');
  IF (Stats.Selected_algorithm = New_algo) THEN
    writeln(F,'No.    Time          Exp_traffic   Gen_traffic   %_use      Int_delay    N_int_delay
               Total_int_delay')
  ELSE
    writeln(F,'No.    Time          Exp_traffic   Gen_traffic   Data_pkt_Rx   %_use      Int_delay');

  FOR i := 1 TO No_of_intervals DO
    BEGIN
      write(F,i:3);
      write(F,Stats.Interval_Time[i]:10:4);
      write(F,Stats.Total_exp_traffic[i]:14);
      write(F,Stats.Total_gen_traffic[i]:14);
      IF (Stats.Selected_algorithm <> New_algo) THEN write(F,Stats.Interval_pkts_Rx[i]:14);
      writeln(F,Stats.Average_percentage_use[i]:12:2);
      write(F,Stats.Interval_pkt_delays[i]:13:4);
      IF (Stats.Sel_algorithm = New_algo)
      THEN writeln(F,Stats.Interval_N_pkt_delays[i]:13:4,Stats.Total_interval_delays[i]:13:4);
      writeln(F);
    END;


{ Write out the routing results if necessary }
{-------------------------------------------}
  IF (Stats.selected_algorithm = New_algo)
  THEN
    BEGIN
      writeln(F);
      writeln(F,'Packets received related to the originating interval');
      writeln(F,'-----------------------------------------------------');
      writeln(F,'No.    Time          Routing pkts Rx       Data pkts Rx        % Routing pkts');

      FOR i := 1 TO No_of_intervals DO
        BEGIN
          write(F,i:3);
          write(F,Stats.Interval_Time[i]:10:4);
          write(F,Stats.Interval_R_pkts_Rx[i]:14);
          write(F,Stats.Interval_pkts_Rx[i]:18);
          write(F,Stats.Percentage_R_pkts_Rx[i]:22:2);
          writeln(F);
        END;
      writeln(F);
    END; { If }

{ Write out delay distribution }
{------------------------------}
  writeln(F);
  writeln(F,'Delay distribution');
  writeln(F,'------------------');
```

```
    Max_no_of_delays := (1 + TRUNC(Stats.Max_delay * 10));

    FOR i := 1 TO Max_no_of_delays DO
      BEGIN
        write(F,Stats.Delay_distribution[i]:5);
        IF (i/15 = TRUNC(i/15)) THEN writeln(F);
      END;
    writeln(F); writeln(F);

  { Write out overall average results }
  {-----------------------------------}
    writeln(F);
    writeln(F,'Overall average results');
    writeln(F,'-----------------------');
    writeln(F,'Duration of newly generated arrivals = ',Duration:10:4);
    writeln(F,'Total duration of simulation         = ',Stats.End_time:10:4);
    writeln(F,'Total average packet delay           = ',Stats.overall_message_delay:10:4);
    writeln(F,'Delay per packet transmitted         = ',Stats.Delay_per_pkt_gen:10:4);
    writeln(F,'Total average delay per hop          = ',Stats.Total_hop_delay:10:4);
    writeln(F,'Maximum packet delay                 = ',Stats.Max_delay:10:4);
    writeln(F,'Total number of packets generated    = ',Stats.Total_pkts_generated:10);
    writeln(F,'Total number of packets discarded    = ',Stats.Total_pkts_discarded:10);
    writeln(F,'Total number of hops (pkts)          = ',Stats.Total_pkts_Tx:10);
    writeln(F,'Total number of blocked packets      = ',Stats.No_of_pkts_blocked:10);
    writeln(F,'Percentage of blocked packets/hop    = ',Stats.Percentage_pkts_blocked:10:2);
    writeln(F,'Packets lost due to no path          = ',Stats.Pkts_lost_due_to_no_link:10);
    writeln(F,'Total % service + I/P queuing time   = ',Stats.Total_service_time:10:2);
    writeln(F,'Total % O/P queuing time             = ',Stats.Total_dept_time:10:2);
    writeln(F,'Total % transmission time            = ',Stats.Total_Tx_time:10:2);
    writeln(F,'Total cost of running network        = ',Stats.Overall_cost:10:2);
    writeln(F);
    CLOSE(F);
  END;

{**************************************************************************}
  BEGIN { Simulation program }

  {***        Initialise data & get data from user and file          ***}
  {-------------------------------------------------------------------}
    Initialise_data(Network,Nodes,Stats_data);
    Get_data_from_file(N,Network,Nodes,Status);
      IF (status <> 0) THEN
        BEGIN
          writeln('ERROR)) Cannot open Output file');
          RETURN;
        END;
    Get_data_from_user(Total_duration,Stats_data.OP_filename);

  {***                 Get initialisation start time                 ***}
  {-------------------------------------------------------------------}
    cal_%get_local_time(init_start_time);
```

```
PROCEDURE Eliminate_loops (VAR Network_defn : networks);
{=========================================================}

  {*****************************************************}
  {*                                                   *}
  {* This procedure is to determine a valid set of blocked *}
  {* paths in the network using a sink tree.           *}
  {* source destination pair.                          *}
  {*                                                   *}
  {* INPUTS:  Variables: The matrix defining the sink tree  *}
  {*          Constants: The network definition        *}
  {* OUTPUTS: Variables: The matrix defining the sink tree  *}
  {*                                                   *}
  {*****************************************************}

  VAR i,j : Node_ID;

  BEGIN
    writeln;
    writeln('Sink tree initialisation');
    writeln('-------------------------');
    FOR j := 1 TO Network_defn.Number_of_nodes DO
      BEGIN
        Find_i(Network_defn,i,1);
        Find_sink_tree(i,1,Network_defn);
        Prune_dead_ends(Network_defn,1);
        Find_more_paths(Network);
        writeln('Sink tree for destination node ',i:3);
        writeln(' Selected source node          ',i:3);
        show_sink_tree(Network_defn,1);
      END;
  END;


PROCEDURE Find_Bl_j (i : Node_ID; Network_defn : Networks;
                                  VAR New_Bl   : All_link_sets);
{=============================================================}

  {*****************************************************}
  {*                                                   *}
  {* The procedure is used to determine the blocked path *}
  {* set for a particular node i and any destination  node *}
  {* j.                                                *}
  {*                                                   *}
  {* INPUTS:  Constant: The node i                     *}
  {*          Constant: The  network  defn,  including the *}
  {*                    blocked path set.              *}
  {* OUTPUTS: Variables: The set Bl[j]                 *}
  {*                                                   *}
  {*****************************************************}
```

```
    VAR n, j : Node_ID;

    BEGIN

      FOR j := 1 TO Network_defn.Number_of_nodes DO
        BEGIN

            { Set blocked paths to the empty set }
            {-------------------------------------}
            New_Bl[j] := [];

            { Scan sink tree for blocked paths }
            {----------------------------------}
            FOR n := 1 TO Network_defn.Number_of_nodes DO
              IF (Network_defn.Sink_tree[i,n,j] = Not_connected)
              THEN
                BEGIN
                  writeln('Node ',i:3,'  Dest ',j:3,'  Block ',n:3);
                  New_Bl[j] := New_Bl[j] + [n];
                END;
        END
    END;


PROCEDURE Find_All_Bl_j (Network_defn      : Networks;
                         VAR All_node_data : All_nodes);
{=========================================================}

  {*****************************************************************}
  {*                                                               *}
  {* The purpose of this procedure is to define the set of         *}
  {* blocked paths  for all the nodes in the network.              *}
  {*                                                               *}
  {* INPUTS:  Variable: The definition of the network.             *}
  {* OUTPUTS: Variable: The  set of blocked paths and node         *}
  {*                    data.                                      *}
  {*                                                               *}
  {*****************************************************************}

  VAR node_i : Node_ID;

  BEGIN
    FOR Node_i := 1 TO Network_defn.Number_of_nodes DO
      Find_Bl_j(Node_i,Network_defn,All_node_data[node_i].Routing_data.
                                                  Adpt_data.Bl);
  END;
```

D.5 Procedures required for transferring routing data between in the network

```
{*********************************************************}
{*                                                       *}
{* Network Routing Simulation Program                    *}
{* ==================================                    *}
{*                                                       *}
{* Name:        'Transmit.pas'                           *}
{*                                                       *}
{* Description: The purpose of the procedures below is to *}
{*              transfer routing information from one node *}
{*              to a neighbouring node.  Piggy-backing is *}
{*              implemented.                             *}
{*                                                       *}
{* Version:     1.0                                      *}
{*                                                       *}
{* Author:      Marc Leon Francois                       *}
{*                                                       *}
{* Date:        5 January 1989                           *}
{*                                                       *}
{*********************************************************}


PROCEDURE Search_event_Q (i, k     : Node_IX; Front_of_Q      : Link;
                          VAR Temp : Link;    VAR Data_pkt_found : BOOLEAN);
{=======================================================================}

  {*********************************************************}
  {*                                                       *}
  {* The function of this process is to search the event *}
  {* queue to determine if a data packet exists in the *}
  {* output queue of node i, neighbouring node k.          *}
  {*                                                       *}
  {* INPUTS:  i, The current node number.                  *}
  {*          k, The neighbouring node number.             *}
  {*             The pointer to the front of the event queue. *}
  {* OUTPUTS: Pointer to the event with the data packet.   *}
  {*          Data packet found is true if a data packet *}
  {*           was found, else it is set to false.         *}
  {*                                                       *}
  {*********************************************************}

  VAR  Event_number : INTEGER32;

  BEGIN
    Temp           := Front_of_Q;
    Data_pkt_found := FALSE;
    Event_number   := 0;
```

```
    WHILE ((Temp^.Sentinal <> TRUE) AND (Data_pkt_found = FALSE)) DO
      BEGIN
        IF ((Temp^.Name = Departure)     AND (Temp^.Node_No = i)
          AND (Temp^.Output_node_No = k) AND (Temp^.Packet.Control_data = data))
        THEN Data_pkt_found := TRUE
        ELSE Temp := Temp^.next_event;
        Event_number := Event_number + 1;
      END;
      IF (data_pkt_found = TRUE) THEN writeln('Piggy back packet found at event ',Event_number:5);
    END;

PROCEDURE Search_input_Q (i             : Node_ID; Foreign_node : Node_ID;
                         Front_of_Q : Link;
                         VAR Temp   : Link;  VAR Data_pkt_found : BOOLEAN);
{==========================================================================}

  {*************************************************************}
  {*                                                           *}
  {* The function of this process is  to search  the event  *}
  {* queue  to  determine if  a data packet exists  in the  *}
  {* output queue of node i, neighbouring node k.              *}
  {*                                                           *}
  {* INPUTS:  i, The current node number.                      *}
  {*          k, The neighbouring node number.                 *}
  {*          The pointer to the front of the event queue.     *}
  {* OUTPUTS: Pointer to the event with the data packet,       *}
  {*          Data packet  found is true if a  data  packet    *}
  {*           was found, else it is set to false.             *}
  {*                                                           *}
  {*************************************************************}

  BEGIN
    Temp          := Front_of_Q;
    Data_pkt_found := FALSE;

    WHILE ((Temp^.Sentinal <> TRUE) AND (Data_pkt_found = FALSE)) DO
      BEGIN

        IF ((Temp^.Name = Service)  AND (Temp^.Node_No = i)
          AND (Temp^.Packet.Control_data = data))
        THEN Data_pkt_found := TRUE
        ELSE Temp := Temp^.next_event;

      END;
    END;
```

```
PROCEDURE Update_data_pkt (Temp       : Link; New_R_previous_node : Node_ID;
                          New_dD_df  : REAL; New_dDt_Dr          : dD_Dr_info;
                                             New_j_dDt_dr        : Node_ID);
{=======================================================================}

   {*****************************************************}
   {*                                                   *}
   {* The function of this process, is when  a data packet *}
   {* has  been found on  which routing data may  be piggy- *}
   {* backed the routi.. data 's added to the data packet.  *}
   {*                                                   *}
   {* INPUTS:  Pointer  to the event  containing  the  data *}
   {*            packet.                                *}
   {*          The new  routing  infomation to be  added to *}
   {*            the data packet.                       *
   {*****************************************************}

   BEGIN
     WITH Temp^.Packet DO
       BEGIN
         Control_data    := Data_and_ctrl;
         dD_df           := New_dD_df;
         dDt_dr          := New_dDt_dr;
         j_dDt_dr        := New_j_dDt_dr; ( = 0 if no valid dDt/dr)
         R_previous_node := New_R_previous_node;
       END;
   END;


PROCEDURE Create_new_R_pkt (Present_time       : REAL;
                            Duration           : REAL;
                            New_dD_dr          : dD_dr_info;
                            dD_dr_dest_node_No : Node_ID;
                            New_dD_df          : REAL;
                            Node_number        : Node_ID;
                            Current_OP_No      : Node_ID;
                            k                  : Node_ID;
                            VAR Node_status    : Node;
                            VAR Q_front,Q_back : Link;
                            VAR Length_of_Q    : INTEGER32;
                            VAR D_pkts_in_Q    : INTEGER32);
{=======================================================}

   {*****************************************************}
   {*                                                   *}
   {* The  function of this  process is simply to  create a *}
   {* new seperate  routing packet if no  data packet  onto *}
   {* the routing packet could be piggy backed was found.   *}
   {*                                                   *}
   {* INPUTS:  Variables: The present time.             *}
   {*                     The duration of the simulation.  *}
   {*                     The new value for dD_dr.      *}
```

```
{*                      The destination  node number  for *}
{*                        which d0_dr was created.        *}
{*                      The new rate of d0/d1.            *}
{*                      The current node number.          *}
{*                      The number of the  node for which *}
{*                        the value of d0_dr was calc.    *}
{*          Pointers:   The front  and back  ptrs. to the *}
{*                      event queue.                      *}
{*          Variable:   The length of the event queue.    *}
{*          Variables:  The status of the node.           *}
{* OUTPUTS: Pointers:   The front  and back  ptrs. to the *}
{*                      event queue.                      *}
{*          Variables:  This status of the node.          *}
{*          Variable:   The length of the event queue.    *}
{*          Variable:   The number of data packets in the *}
{*                      event queue.                      *}
{*                                                        *}
{**********************************************************}

VAR  Dept_time      : REAL;
     Bits_per_sec   : REAL;
     New_event      : Event_Info;
     Temp_ptr       : Link;
     New_event_ptr  : Link;
     Insert         : BOOLEAN;

BEGIN
  IF (Node_status.Output_Q_length[k] < Node_status.Max_output_Q_length)
  THEN writeln('Routing packet blocked on output path')
  ELSE
    BEGIN

    { Calculate the departure time of the routing packet }
    {-----------------------------------------------------}
      Bits_per_sec := 1000 * Node_status.Routing_data.Link_capacities[Node_number,k]
                           / Node_status.Bits_per_packet;
      Dept_time    := Present_time + (Node_status.Output_Q_length[k]
                           /Bits_per_sec) + 1 * Delta1;

    { Update packet and add it to the event queue }
    {---------------------------------------------}
      WRITE(Node_status.Output_Q_length[k]);
      Update_an_event(Duration,Dept_time,Node_number,Node_number,k,Node_number,
                      0,control,departure,0,Dept_time,New_algo,New_event);
      New_event.Output_node_No          := k;
      New_event.Previous_arrival_time   := Dept_time;
      New_event.Packet.d0t_dr           := New_d0_dr;
      New_event.Packet.j_d0t_dr         := d0_dr_dest_node_No;
      New_event.Packet.d0_df            := New_df_d1;
      New_event.Packet.Previous_service_time := Present_time;
      New_event.Packet.Time_created     := Present_time;
      New_event.Packet.R_previous_node  := Node_number;
```

```
        New(New_event_ptr);
        New_event_ptr^ := New_event;
        Find_insert_pos(Dept_time,D_front_D_back,Temp_ptr,Insert);
        Insert_event(Temp_ptr,New_event_ptr,Length_of_D_D_ptrs_in_D,Insert);
    END; { else }
END;


PROCEDURE Transmit_D_data (Present_time      : REAL;
                           Duration          : REAL;
                           New_dD_dr         : dD_dr_info;
                           dD_dr_dest_node_No : Node_ID;
                           New_dD_df         : REAL;
                           Node_number       : Node_ID;
                           Current_DP_No     : Node_ID;
                           k                 : Node_ID;
                       VAR Node_status       : Node;
                       VAR D_front_D_back    : Link;
                       VAR Length_of_D       : INTEGER32;
                       VAR D_ptrs_in_D       : INTEGER32);
{=========================================================================}

  {*************************************************************************}
  {*                                                                       *}
  {* The  function of this process is to transmit  routing                 *}
  {* data to neighbouring node. First the procedure checks                 *}
  {* if the packet can be piggy-backed,  if not a separate                 *}
  {* routing packet is generated.                                          *}
  {*                                                                       *}
  {* INPUTS:  Variables: The present time.                                 *}
  {*                     The duration of the simulation.                   *}
  {*                     The new value for dD_dr.                          *}
  {*                     The destination  node number  for                 *}
  {*                      which dD_dr was created.                         *}
  {*                     The new vale of dD/df.                            *}
  {*                     The current node number.                          *}
  {*                     The number of the  node for which                 *}
  {*                      the value of dD_dr was calc.                     *}
  {*           Pointers: The front  and back  ptrs. to the                  *}
  {*                      event queue.                                     *}
  {*           Variable: The length of the event queue.                    *}
  {*           Variables: The status of the node.                          *}
  {* OUTPUTS: Pointers: The front  and back  ptrs. to the                  *}
  {*                      event queue.                                     *}
  {*           Variables; The status of the node.                          *}
  {*           Variable: The length of the event queue.                    *}
  {*           Variable: The number of data packets in the                 *}
  {*                      event queue.                                     *}
  {*                                                                       *}
  {*************************************************************************}
```

```
VAR Piggy_back_pkt_found : BOOLEAN;
    Temp_ptr             : Link;

BEGIN
  Piggy_back_pkt_found := FALSE;
  IF (Node_status.Output_Q_length[k] > 0)
  THEN Search_event_Q(Node_number,k,Q_front,Temp_ptr,Piggy_back_pkt_found);

  IF (Piggy_back_pkt_found = TRUE)
  THEN Update_data_pkt(Temp_ptr,Node_number,New_dD_dr,New_dD_dr,dD_dr_dest_node_No)
  ELSE
    BEGIN
      IF (Node_status.Output_Q_length[k] = Node_status.Max_Output_Q_length)
      THEN writeln('Routing packet blocked on output path')
      ELSE Create_new_D_pkt(Present_time,Duration,New_dD_dr,dD_dr_dest_node_No,
                            New_dD_dr,Node_number,Current_DP_No,k,Node_status,
                            Q_front,Q_back,Length_of_Q,D_pkts_in_Q);
    END;
END;


PROCEDURE Transmit_new_dDt_dr (Present_time      : REAL;
                               Duration          : REAL;
                               New_dD_dr         : dD_dr_Info;
                               dD_dr_dest_node_No : Node_ID;
                               Node_number       : Node_ID;
                               Current_DP_No     : Node_ID;
                               VAR Node_status   : Node;
                               VAR Q_front,Q_back : Link;
                               VAR Length_of_Q   : INTEGER32;
                               VAR D_pkts_in_Q   : INTEGER32);
{=============================================================}

  {*********************************************************
  (* The function of this procedure is after having calc- *)
  (* ulated a new value for dD_dr from the 'Update routing *)
  (* data' procedure, the value is transmitted to all     *)
  (* neighbouring nodes except the one for which the value *)
  (* of dD_dr was calculated. Using this procedure a      *)
  (* routing packets are generated and inserted ont. the  *)
  (* event queue.                                         *)
  (* INPUTS:  Variables: The present time.                *)
  (*                     The duration of the simulation.  *)
  (*                     The new value for dD_dr.         *)
  (*                     The destination node number for  *)
  (*                       which dD_dr was created.       *)
  (*                     The current node number.         *)
  (*                     The number of the node for which *)
  (*                       the value of dD_dr was calc.   *)
  (*          Pointers:  The front and back ptrs. to the  *)
  (*                       event queue.                   *)
  (*          Variable:  The length of the event queue.   *)
```

APPENDIX D

```
{* OUTPUTS: Variables: The statistic variables.          *}
{*          Pointers:  The front  and back  ptrs. to the *}
{*                     event queue.                      *}
{*          Variable:  The length of the event queue.    *}
{*****************************************************}

VAR  k                : Node_ID;

BEGIN
  FOR k := 1 TO Node_status.Routing_data.Number_of_nodes DO
    BEGIN
      IF ((Node_status.Routing_data.Link_capacities[Node_number,k] <> 0)
          AND (Current_OP_No <> k)
          AND (OD_dr_dest_node_No <> k)
          AND (NOT(k IN Node_status.Routing_data.Adpt_data.Bit[OD_dr_dest_node_No])))
        THEN
          BEGIN
            Transmit_D_data(Present_time,Duration,New_OD_dr,OD_dr_dest_node_No,
                            Node_status.Routing_data,Adpt_data.Inc_delays[k],OD_dr_Ei,
                            Node_Number,Current_OP_No,k,Node_status,
                            Q_front,Q_back,Length_of_Q,D_pkts_in_Q);
          END
    END; { for k }
END;


PROCEDURE Transmit_OD_Of (Present_time       : REAL;
                          Duration           : REAL;
                          New_OD_Of          : REAL;
                          OD_Of_dest_node_No : Node_ID;
                          Node_number        : Node_ID;
                          VAR Node_status    : Nodes;
                          VAR Q_front,Q_back : Link;
                          VAR Length_of_Q    : INTEGER32;
                          VAR D_pkts_in_Q    : INTEGER32);
{=========================================================}

  {#################################################}
  {*                                                 *}
  {* The function of this procedure is after having calc- *}
  {* ulated a new value for OD_Of the value is transmitted *}
  {* to the appropriate node.                        *}
  {*                                                 *}
  {* INPUTS:  Variables: The present time.           *}
  {*                     The duration of the simulation.  *}
  {*                     The new value for OD_Of.    *}
  {*                     The destination  node number  for *}
  {*                      which OD_Of was created.   *}
  {*                     The current node number.    *}
  {*          Pointers:  The front  and back  ptrs. to the *}
  {*                     event queue.                *}
  {*          Variable:  The length of the event queue.  *}
```

```
(* OUTPUTS: Variables: The statistic variables.          *)
(*          Pointers:  The front  and back  ptrs. to the *)
(*                     event queue.                      *)
(*          Variable:  The length of the event queue.    *)
(*                                                       *)
(*********************************************************)

VAR  Temp_dD_dr : dD_dr_info;

BEGIN
  Temp_dD_dr.dD_dr := 0;  Temp_dD_dr.Tag := FALSE;

  IF (Node_status.Routing_data.Link_capacities[Node_number,dD_df_dest_node_no] <> 0)
  THEN
    BEGIN
      Transmit_D_data(Present_time,Duration,Temp_dD_dr,0,New_dD_df,Node_number,
                      dD_df.dest_node_No,dD_df.dest_node_No,Node_status,
                      D_front,D_back,Length_of_D_pkts_in_Q);

    END;
END;


{=============================================================================}
```

```
PROCEDURE Receive_routing_data (Event_defn      : Event_info;
                                VAR Node_status : Node     );
{=============================================================}


  {*************************************************************}
  {*                                                           *}
  {* The function of this procedure is to process the  in-     *}
  {* coming routing data to a node.                            *}
  {*                                                           *}
  {* INPUTS:  Event defn:  The event definition containing     *}
  {*                       the the data  contained  within     *}
  {*                       the newly arrived packet.           *}
  {*          Node Status: The current states of the  node at  *}
  {*                       which the packet arrived.           *}
  {* OUTPUTS: Node Status: The updated routing  tables  at     *}
  {*                       the node.                           *}
  {*                                                           *}
  {*************************************************************}

  VAR  new_k, current_j : Node_Ty;

  BEGIN

    New_k      := Event_defn.Packet.R_Previous_node;
    current_j := Event_defn.Packet.j_dDi_dr;

    { Get new dD/dt from the packet }
    {------------------------------}
    IF (Current_j = 0) THEN
      BEGIN
        Node_status.Routing_data.Adpt_data.dD_dt_ik[Event_defn.Packet.R_Previous_node]
            := Event_defn.Packet.dD_dt;
        IF (NOT(new_k IN[Node_status.Routing_data.Adpt_data.dD_dfs_Rx]))
        THEN Node_status.Routing_data.Adpt_data.dD_dfs_Rx :=
             Node_status.Routing_data.Adpt_data.dD_dfs_Rx + [new_k];
      END;

    { Check if Packet contained new dD/dr information }
    {-------------------------------------------------}
    IF (Current_j <> 0)
    THEN
      BEGIN
        Node_status.Routing_data.Adpt_data.
          dD_dr_data[Event_defn.Packet.R_Previous_node,Current_j]
            := Event_defn.Packet.dDi_dr;
        IF (NOT(new_k IN[Node_status.Routing_data.Adpt_data.R_data_Rx[Current_j]]))
        THEN Node_status.Routing_data.Adpt_data.R_data_Rx[Current_j] :=
             Node_status.Routing_data.Adpt_data.R_data_Rx[Current_j] + [new_k];
```

```
            { Else k not element of the blocked paths }
            {-----------------------------------------}
              ELSE
                BEGIN
                  Alk_j := R_data.adpt_data.dD_df_ik[k]
                           + R_data.adpt_data.dD_dr_data[k,j].dD_dr - Gain;


                  IF (R_data.Traffic_data[j].Total_traffic <> 0)
                  THEN R_data.adpt_data.deltak[k,j] := Min(R_data.Phi[k,j],100*alpha*Alk_j/
                                                       R_data.Traffic_data[j].Total_traffic)
                  ELSE R_data.Adpt_data.deltak[k,j] := 0;
                END; {else}
          END; {For k}

        FOR k := 1 TO N DO
          IF (k <> kmin)
          THEN
            BEGIN
              IF (R_data.Phi[k,j] < R_data.adpt_data.deltak[k,j])
              THEN
                BEGIN
                  R_data.adpt_data.deltak[k,j] := R_data.Phi[k,j];
                  R_data.Phi[k,j]              := 0;
                END
              ELSE R_data.Phi[k,j] := R_data.Phi[k,j] - R_data.adpt_data.deltak[k,j];
              Total_delta := Total_delta + R_data.adpt_data.deltak[k,j];
              Total_phi   := Total_phi + R_data.phi[k,j];
            END;

        k                 := kmin;
        R_data.phi[k,j] := R_data.phi[k,j] + Total_delta;
        Total_phi         := Total_phi + R_data.phi[k,j];

        { If total phi (accurate to 4 decimal places) = 1 then terminate loop }
        {---------------------------------------------------------------------}
          IF (TRUNC(10000*Total_phi) = 10000) THEN Finished := TRUE;

        { If total phi <> 1 then correct error }
        {--------------------------------------}
          IF (Finished = FALSE)
          THEN
            BEGIN
              delta_phi_error := 1.00000 - Total_phi;
                R_data.Phi[R_data.shortest_path_node[j],j] :=
                R_data.Phi[R_data.shortest_path_node[j],j] + delta_phi_error;
            END;
      END; {procedure}
```

```
{* INPUTS:  Variables: The number of nodes in the network *}
{*                     The node number where the routing *}
{*                     data is to be updated (i).        *}
{*                     The destination node number (j).  *}
{* OUTPUTS: Variables: The routing data.                 *}
{*                                                       *}
{*******************************************************}

VAR  k,n, kmin                      : Node_ID;
     dit_j, Dim_plus_d0_dr, Total_delta,
     New_d0_dr, Dmin, Total_phi     : REAL;
     delta_phi_error                : REAL;
     Finished                       : BOOLEAN;

BEGIN
   Total_phi := 0; Finished := FALSE; Total_delta := 0;

  { Calculate minimum d0/dl + d0/dr and Kmin }
  {-----------------------------------------}
    Dmin := infinity;
    FOR n := 1 TO N DO
      IF (NOT(n IN R_data.Adpt_data.B(i,j)))
      THEN
        BEGIN
          Dim_plus_d0_dr := R_data.adpt_data.d0_df_ik[n]
                          + R_data.adpt_data.d0_dr_data(n,j).d0_dr;
          IF (Dim_plus_d0_dr < Dmin)
          THEN
            BEGIN
              Dmin := Dim_plus_d0_dr;
              kmin := nc.
            END;
        END; {for n}


  { Perform algorithm to update phi's }
  {-----------------------------------}
    FOR k := 1 TO N DO
      BEGIN

        { If element of the blocked paths }
        {---------------------------------}
          IF (k IN R_data.adpt_data.B(i,j))
          THEN
            BEGIN
              R_data.phi[k,j]              := 0;
              R_data.adpt_data.deltak[k,j] := 0;
            END
```

```
    { Predict future traffic }
    {-----------------------}
    { New_traffic.Tn_2          := New_traffic.Tn_1;
      New_traffic.Tn_1          := New_traffic.Total_traffic;
      New_traffic.Total_traffic := 2 * New_traffic.Tn_1 - New_traffic.Tn_2;}
    IF (New_traffic.No_of_pkts = Max_trf_pkts)
    THEN
      BEGIN
        New_traffic.No_of_pkts  := 0;
        New_traffic.Update_time := Current_time;
      END;
  END;

PROCEDURE Calculate_new_dD_dr (N, j, i : Node_ID; VAR R_data : Routing_Info);
{=========================================================================}

  {#########################################################}
  {#                                                       #}
  {# The function of this procedure is to calculte the new #}
  {# values for dD/dr_ik(j).                               #}
  {#                                                       #}
  {#########################################################}

  VAR  k          : Node_ID;
       New_dD_dr  : REAL;

  BEGIN

    { Calculate new dD/dr(k,j) }
    {-------------------------}
        New_dD_dr := 0;
        FOR k := 1 TO N DO
          IF (R_data.Link_capacities[i,k] > 0)
          THEN  New_dD_dr := New_dD_dr + (R_data.Adpt_data.dD_di_likE
                                + R_data.Adpt_data.dD_dr_data[k,j].dD_dr) * a_rate.Phi[k,j];

        R_data.adpt_data.dD_dr_data[i,j].dD_dr := New_dD_dr;
        Calc_blocked_paths(R_data.Number_of_nod...s,i,j,R_data);
  END;

PROCEDURE Update_Routing_Data (N, i, j    : Node_ID;
                               VAR R_data : Routing_Info);
{=========================================================}

  {#########################################################}
  {#                                                       #}
  {# The function of this procedure is to implement  an #}
  {# adaptive routing algorithm based on  R. Gallager's #}
  {# routing algorithm.                                    #}
  {#                                                       #}
```

```
            BEGIN
              WITH R_data.Adpt_data DO Temp_data := Alpha * (d0_of_ik[a]
                                  + d0_dr_data[a,j]-d0_dr - d0_dr_data[i,j].d2_dr);
              IF (R_data.Traffic_data[j].Total_traffic <> 0)
              THEN Temp_data := Temp_data/R_data.Traffic_data[j].Total_traffic
              ELSE Temp_data := infinity;

              IF (R_data.Phi[a,j] >= Temp_data)
              THEN
                BEGIN
                  Tag_pkt := TRUE;
                END;
            END; {if Phi}

          { Check if incoming packet indicated a improper path }
          {-------------------------------------------------}
            IF (R_data.Adpt_data.d0_dr_data[a,j].Tag = TRUE) THEN Tag_pkt := TRUE;

        END; {for a}

      R_data.Adpt_data.d0_dr_data[i,j].Tag := Tag_pkt;

    END; { Procedure }


  PROCEDURE Calc_traffic (Current_Time    : REAL;
                          Bits_per_pkt    : INTEGER;
                          VAR New_traffic : Traffic_status);
{==========================================================}

    {*****************************************************}
    {*                                                   *}
    {* The function of this procedure is to calculate the *}
    {* total traffic IT[i,j] at node i to a particular dest- *}
    {* ination node j.                                   *}
    {*                                                   *}
    {* INPUTS:  The current time.                        *}
    {*          The number of bits per packet.           *}
    {*          The traffic data                         *}
    {* OUTPUTS: The new traffic data.                    *}
    {*                                                   *}
    {*****************************************************}

    BEGIN
      INC(New_traffic.No_of_pkts);
      IF (Current_time <> New_traffic.Update_time)
      THEN New_traffic.Total_traffic := ((New_traffic.No_of_pkts (* Bits_per_pkt*))
                                        /(Current_time - New_traffic.Update_time));
```