

12-1-1983

The system simulation for communication network

Taksin Plabjang
Atlanta University

Follow this and additional works at: <http://digitalcommons.auctr.edu/dissertations>



Part of the [Computer Sciences Commons](#), and the [Mathematics Commons](#)

Recommended Citation

Plabjang, Taksin, "The system simulation for communication network" (1983). *ETD Collection for AUC Robert W. Woodruff Library*. Paper 2584.

This Thesis is brought to you for free and open access by DigitalCommons@Robert W. Woodruff Library, Atlanta University Center. It has been accepted for inclusion in ETD Collection for AUC Robert W. Woodruff Library by an authorized administrator of DigitalCommons@Robert W. Woodruff Library, Atlanta University Center. For more information, please contact cwiseman@auctr.edu.

THE SYSTEM SIMULATION

FOR

COMMUNICATION NETWORK

A THESIS

SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY

FOR FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF MASTER OF SCIENCE

BY

TAKSIN PLABJANG

DEPARTMENT OF MATHEMATICAL SCIENCES

ATLANTA, GEORGIA

DECEMBER, 1983

R-1V

P-121

TABLE OF CONTENTS

LIST OF FIGURES	i
LIST OF TABLES	ii
ACKNOWLEDGMENTS	iii
ABSTRACT	iv
Chapter	
1. INTRODUCTION	1
2. GENERAL SYSTEM SIMULATION	4
3. RANDOM NUMBER	10
4. MANAGING QUEUE	12
5. COMMUNICATION NETWORK SYSTEM	15
6. STATIC DESCRIPTION OF THE SIMULATION NETWORK	19
7. DYNAMIC DESCRIPTION OF THE SIMULATION NETWORK	30
8. DOCUMENTATION FOR SIMULATION NETWORK PROGRAM	38
9. INPUT AND OUTPUT FORMAT	85
10. ANALYSIS RESULTS OF SIMULATION NETWORK PROGRAM	94
APPENDIX A FLOW CHART OF FUNCTIONS MANAGING QUEUE	102
APPENDIX B TESTING RANDOM NUMBER	106
APPENDIX C GLOSSARY	113
APPENDIX D SYMBOL REPRESENTATION	117
BIBLIOGRAPHIES	119

LIST OF FIGURES

Figure

1. Sample of Communication Network	16
2. Sample of Simulation Network Model	18
3. A Network Consisting of 2 Nodes and a Link	19
4. A Network Consisting of 4 Nodes and 4 Links	20
5. Symbol and the Parameter for Originator Node	23
6. Symbol and the Parameter for Delay Node	26
7. Symbol for Destroyer Node	27
8. Simulation Network Model	29
9. Model for Simulation Network Activity	30
10. Events and States in the System	31
11. List of Events in Event Queue	32
12. Model of Simulation Program Usage	34
13. The Event Scheduling Process	37
14. Procedure Dependence for Simulation Network	37
15. The Structure of the Event Queue	41
16. The Structure of the Delay Queue	43
17. Flow Chart for the Simulation Network Program ...	59
18. Format of Input Data File	87
19. Flow Chart of Procedure Inserteventqueue	102
20. Flow Chart of Procedure Getnextevent	103
21. Flow Chart of Procedure Insert	104
22. Flow Chart of Procedure Delete	105

LIST OF TABLES

Table

1. List of 100 Random Number in Length 0..1	11
2. Parameter for Originator and Delay Node in Network Model	28
3. Coding Main Program List of Simulation Network ...	57
4. Statistical Table for Simulation Network	89
5. Example of Debug Output for Action Listing	90
6. Example of List of Events in Event Queue	92
7. Example of Summary Report of Simulation Network ..	93
8. Data Describing a Simulation Network System	94
9. Results of a Simulation Network Model	96
10. Data Describing another Simulation Network System.	98
11. Results of another Simulation Network Model	99
12. Sequence of 100 Random Numbers, Seed = 0	107
13. Sequence of 100 Random Numbers, Seed = 4	107
14. List of Event Times, Seed=0, Low=2.5, High=4.5 ...	108
15. List of Event Times, Seed=4, Low=2.5, High=4.5 ...	109
16. Results of Simulation Network, Seed = 0, Maxtime = 100	110
17. Results of Simulation Network, Seed = 4, Maxtime = 100	111

ACKNOWLEDGEMENTS

I wish to express my deepest appreciation, especially for my advisor, Dr. Bennett Setzer. This thesis could not be accomplished without his support and patience. Furthermore, I would like to extend my respect to the following Mathematical Department instructors :

Dr. Benjamin J. Martin

Dr. Johnny L. Houston

Dr. Nazir A. Warsi

Dr. Grover Simmons

Dr. Ronald Biggers

Dr. Arthur M. Jones

Dr. Henry Gore

Those instructors have been extremely helpful to me for all the time that I have been Atlanta University student.

ABSTRACT

COMPUTER SCIENCE

Taksin Plabjang

B.A. Chulalongkorn University,
Bangkok, Thailand, 1979.

THE SYSTEM SIMULATION FOR COMMUNICATION NETWORK.

Advisor : Dr. Bennett Setzer

Thesis date December, 1983

The advent of computers has made it feasible to approach the study of such complex systems by means of simulation. Computer simulation is a design tool to minimize cost, planning, and implementation. Computer simulation also is a well-known and popular technique for studying the behavior of complex systems. The communication network system is one of the complex systems that should be studied by the simulation model.

In order to gain a fuller understanding of the simulation of communication a network system, the author made this attempt to develop and document his simulation network program.

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

We have developed a simulation program which can model certain kinds of communication networks, namely packet switched trunk level networks. The program can also be used to model other systems, such as student registration at a college. In our program, the model is represented by a network containing three types of nodes. Entities called packets move through the network simulating the activity of the real system. An originator node creates a packet and sends it to other nodes. A destroyer node receives a packet and removes it from the system. The third kind of node, called a delay node, is a single server queue.

A communication network consists of several nodes connected by transmission lines. We model this by representing each communication node by an originator node and a destroyer node. The originator node represents the communication node transmitting a packet generated locally. The destroyer node represents the communication node receiving a packet and removing it from the network. Delay nodes are used to represent delays in transit from one

communication node to another. They can also represent delays within a node through which a packet is simply passing. One goal of a communication network is to minimize the transit time of a packet from transmission until reception. We use the simulation model to provide the good results. By varying parameters and geometry, characteristics of methods to improve the performance of the communication network can be found.

The simulation program is written in PASCAL and runs on the VAX 11/780. Several features of this program are

(a) Input file describes network. This program allows the user to use an input data file to contain data to specify the network. This includes the parameter values of originator nodes and delay nodes.

(b) Diagnostic options. This program has two options for the user to see each event after performing an action and/or all of the events pending in the event queue.

(c) Delays and routing can be randomly generated in this communication network system.

(d) Length of simulation time is specified by the user. This program allows the user to define the maximum simulated time for the run.

The efficiency of this program depends on the number of events that are simulated and the CPU time. For example, in

one run of 50 units simulation time, there were 89 events simulated in that length of time with a CPU time of 0.56 seconds. This means the program simulated 150 events per CPU second.

CHAPTER 2

GENERAL SYSTEM SIMULATION

CHAPTER 2

GENERAL SYSTEM SIMULATION

There are many complex systems which are difficult to study such as (1) communication networks, (2) manufacturing systems, (3) registration process at a college. To minimize the high cost of such systems, planning and implementation must be as efficient as possible. One technique of planning which yields results quickly and at relatively low cost is known as simulation. Simulation makes it possible to test a system before deciding to invest in a corresponding real system.

Computer simulation is the process of conducting experiments on a computer model of a dynamic system. The immediate purpose of these experiments is to observe the behavior of a system under a given set of assumptions, conditions and parameter values. In this chapter, we show how to use system simulation to study the behavior of a communication network system. We use discrete simulation modeling to model our network system. "In discrete simulation, the state of the system can change only at event times. Since the state of the system remains constant

between event times, a complete dynamic of the state of the system can be obtained by advancing simulated time from one event to the next."¹

We use a queue of scheduled events prioritized by time to drive the simulation clock. Next, we will be comparing a definition for various parts of a simulation experiment by William G. Bulgren, related to our project.

" A system is defined as an aggregation or assemblage of objects joined in some regular interaction or interdependence."²

For example, a communication network consists of several nodes connected by transmission lines. As another example, our simulation network consists of nodes of several types connected by paths. There are originator nodes to create packets which move along the paths through the system, delay nodes representing process delays, and destroyer nodes for removing packets from the system.

" An entity is an object of interest in a system."³

In our communication network system we have the

¹ A. Alan, B. Pritsker, and Claude Dennis, Introduction to Simulation and SLAM (New York : A Halsted Press Book, John Wiley and Sons, 1979), p.64

² William G. Bulgren, Discrete System Simulation (Englewood Cliffs, New Jersey : Prentice-Hall Inc., 1982), p. 2

³ Ibid.

following entities: nodes, message packets, and transmission lines; In our simulation network system: originator nodes, delay nodes, destroyer nodes, packets, and paths are entities.

" An attribute is a property of an entity."⁴

In our communication network system we have the following attributes for each entity

Nodes : Sending and receiving a packet.

Message packets : Route through system, and time in system.

Transmission lines : Target node of path.

We also have attributes for each entity in our simulation network as follow

Originator node : Distribution of packets originating at this node.

Delay node : Distribution of service times through the node and distribution of exit paths for packets leaving the node.

Destroyer node : No attribute.

Packets : Delay time in system, transit time, and service time.

⁴ William G. Bulgren, Discrete System Simulation (Englewood Cliffs, New Jersey : Prentice-Hall Inc., 1982), p. 3

Paths : Target node of path.

" An activity is a process that cause changes in the system."⁵

The activities in our communication network are sending packets and receiving packets. We also have many activities in our simulation network such as creating a new packet, a packet leaving a delay node, destroying a packet, packet entering a delay queue, and serving a packet waiting in a service queue. An event is a combination of activities that occur in response to some single activity. We have three major events in our simulation network. These are

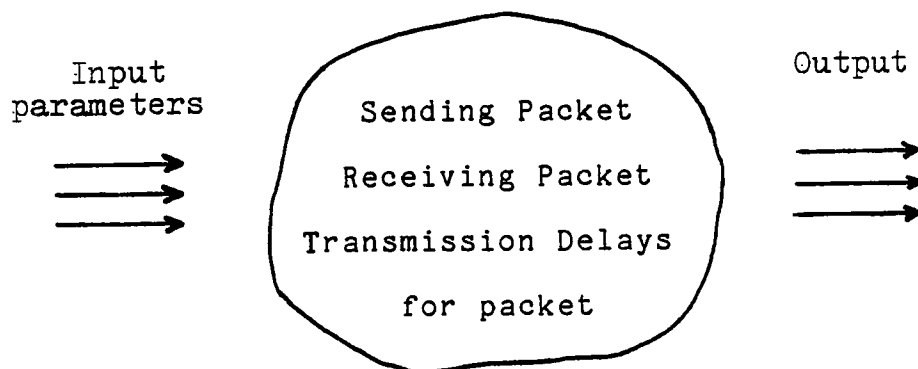
1. Create event. Create a new packet at an originator node. This event includes some of these activities. They are packet entering a delay server and packet entering a delay queue.
2. Leave event. Packet leaves a delay node moving to another delay node or a destroyer node. This event includes some of these activities. They are serving a packet waiting in service queue, packet entering a delay server or packet entering a delay queue, and destroying a packet.
3. Stop event. Stop the simulation.

⁵ William G. Bulgren, Discrete System Simulation (Englewood Cliffs, New Jersey : Prentice-Hall Inc., 1982), p. 3

" A model is a representation of a system to describe the system in sufficient detail for the behavior of the model to provide valid predictions of the behavior of the system."⁶

A communication network consists of several nodes connected by transmission line. The concepts of our communication network system are sending packets, receiving packets, and delays in transmission for the packet. The model for our communication network is represented by a simulation network containing three types of nodes. These communication nodes are represented by originator nodes, destroyer nodes, and sometimes delay nodes. In the case of a particular communication network, an input data file is created which contains the parameter values of originator nodes and delay nodes to describe our communication network system.

Communication Network Model



⁶ William G. Bulgren, Discrete System Simulation (Englewood Cliffs, New Jersey : Prentice-Hall Inc., 1982), p. 4

In our network model we use the input parameter values to compute the creation time distribution for new packets. New packets are sent from originator nodes to delay nodes or destroyer nodes. If a packet is sent to a delay node we also use the input parameter values to compute the delay time distribution in that delay node and also to compute the exitpath from that delay node. But if the packet is sent to a destroyer node, we save the transit time and delay time of the packet for the output of the model.

CHAPTER 3
RANDOM NUMBER

CHAPTER 3
RANDOM NUMBER

In our simulation program, we must take samples from various statistical distributions. We use a pseudo-random number generator to do this. The generator is based on the linear congruence method. This produces a random number using the following formula⁷:

$$\begin{aligned} \text{Seed} &= (\text{multiplier} \times \text{Seed} + \text{increment}) \bmod \text{modulus} \\ \text{Random number} &= \text{Seed}/65536 \end{aligned}$$

We use the following coefficients⁸ to generate random number.

$$\begin{aligned} \text{modulus} &= 2^{16} = 65536 \\ \text{multiplier} &= 25173 \\ \text{increment} &= 13849 \end{aligned}$$

This formula provides a pseudo-random sample in the

⁷ Peter Grogono, Programming in PASCAL (Concordia University, Montreal Inc., 1980), p. 119

⁸ Ibid.

interval [0,1]. Our simulation program allows the user to specify the initial value of the seed for generating random numbers.

For example, Table 3.1 is a list of 100 random numbers generated by using the initial value of seed equal 0.

0.211	0.744	0.476	0.262	0.212	0.936	0.793	0.474	0.480	0.290
0.305	0.871	0.936	0.163	0.560	0.937	0.753	0.167	0.356	0.406
0.352	0.160	0.739	0.263	0.405	0.252	0.005	0.742	0.834	0.802
0.702	0.328	0.810	0.419	0.626	0.754	0.757	0.963	0.825	0.005
0.345	0.543	0.595	0.316	0.497	0.518	0.484	0.171	0.383	0.499
0.287	0.304	0.428	0.344	0.053	0.700	0.301	0.162	0.077	0.593
0.925	0.313	0.906	0.468	0.472	0.407	0.339	0.058	0.364	0.302
0.420	0.349	0.272	0.109	0.448	0.574	0.119	0.186	0.970	0.218
0.076	0.143	0.781	0.015	0.566	0.839	0.929	0.950	0.259	0.384
0.711	0.257	0.079	0.137	0.674	0.420	0.196	0.707	0.614	0.175

Table 3.1 list of 100 random numbers in interval [0,1]

This simulation network program is written in PASCAL, therefore, the largest value of the coefficient modulus is 2^{16} . That means we can generate 65536 random numbers and the sequence repeats. If we use another language such as Assembly language, to write this program, the value of modulus can be rather large to 2^{35} . So, we have more than 65536 random numbers generated.⁹

⁹ Donald E. Knuth, The Art of Computer Programming (Stanford University: Addison-Wesley Publishing Company, Inc., 1969), p. 88

CHAPTER 4

MANAGING QUEUE

CHAPTER 4

MANAGING QUEUE

Most simulation models need functions to manage queues. A queue is a data structure used for dynamic temporary storage of elements. Elements may enter a queue at any time. When an element is removed from queue, various rules can be used to decide which element to select :

- (1) FIFO (First-In-First-Out)
- (2) Priority measure.

In our simulation network model, we have two kinds of queues. They are event queues and delay queues. A delay queue is a FIFO, an event queue is a priority queue based on least time of event occurrence. We use a linked list structure to implement both types of queues. There is one event queue in this system.

Event queue

The event queue contains a list of all events in the network system to be processed and keeps them sorted by the time they are to occur. We use a linear search to find the place for a new event to be inserted into the event queue.

The event queue is a priority queue. It keeps the event list ordered by time. On each simulation cycle, we advance the clock to the next event in the event queue and process that new event. Therefore, the event that has the smallest event time is placed at the head of event queue and the event that has the biggest event time is placed at the tail of event queue. We have two activities involved with the eventqueue. They are to insert in and delete from the event queue. These two activities are handled by two procedures named Inserteventqueue and Getnextevent respectively.

Inserteventqueue : This procedure adds a new event to the event queue, keeping the event list ordered by time field. After inserting a new event into the event queue, a pointer points to the next event in the event queue. A flow chart of this procedure can be found in Appendix A, page 102

Getnextevent : This procedure removes the next event from the front of event queue. After deleting first event from the event queue, a pointer points to the new first event in the event queue. A flow chart of this procedure can be found in Appendix A, page 103

Delay queue

Each delay node in our network system has a queue for packets which are waiting for service. The delay queues are

first-in-first-out (FIFO) . The first packet that waits in a delay queue will be deleted first from the queue. The delay queue also has two activities, insert packet into delay queue and delete packet from delay queue. Again, we have two procedures to handle these activities, called Insert and Delete.

Insert : This procedure inserts a packet at the end of delay queue. After inserting a new packet, a pointer points to nil. A flow chart of this procedure can be found in Appendix A, page 104

Delete : This procedure takes the first packet from a delay queue. After deleting the first packet, a pointer points to the front of the queue. A flow chart of this procedure can be found in Appendix A, page 105

In our simulation network model, both delay queues and the event queue have no limit on the number of packets and number of events that can be stored. This is because we use a linked list structure to implement those queues. Note that, for a given simulation network, the maximum size of the event queue can be predicted as the number of originator nodes plus number of delay nodes plus 1 (stop event). It is convenient, however, for the system to be able to handle a large size network without recompiling.

CHAPTER 5

COMMUNICATION NETWORK SYSTEM

CHAPTER 5

COMMUNICATION NETWORK SYSTEM

A communication network is the system that transfer and processing of discrete units of information within the network. The communication network consists of several nodes connected by transmission lines. Two kinds of communication nodes are

1. Sender node. Node that transmits a packet through the system and
2. Receiver node. Node that receives a packet and removes it from the network.

For example, we can have 3 nodes in a system and the activities of each node are sending, receiving, and passing a packet. Figure 5.1 show a sample of communication network.

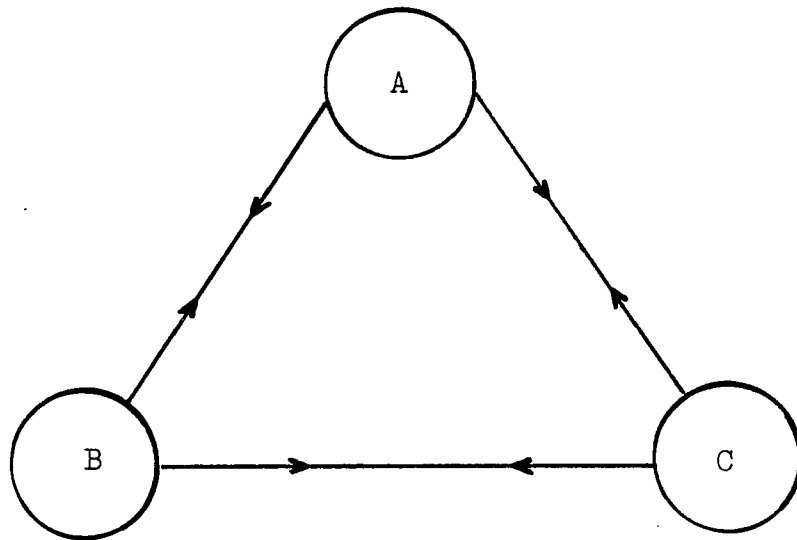


Figure 5.1 Sample of communication network

From Figure 5.1, nodes A,B, and C are both senders and receivers. For example, node A sends packets through the transmission lines to nodes B and C and also receives packets sent from nodes B and C.

Now, we try to study the behavior of a communication network system by using simulation. The model for our communication network is represented by a simulation network. The simulation network contains three types of nodes:

1. Originator node. An originator node represents a sender node. This node generates packets and routes them into the system. An originator node contains three types of variable. Minwait and Maxwait are used to compute next creation time of new packet, Exit represent a target node from originator node.
2. Destroyer node. A destroyer node represents a receiver node. This node destroys a packet from the system.
3. Delay node. A delay node represents a transmission line. This node is used to represent delay in transit from node to node. A delay node contains four types of variable. Holdmin and Holdmax are used to compute service time, Exits represents list of target nodes from delay node, and Exprob represents probabilities of target nodes.

If we model a transmission line by delay node, from Figure 5.1 we can create a simulation network model that is shown in Figure 5.2

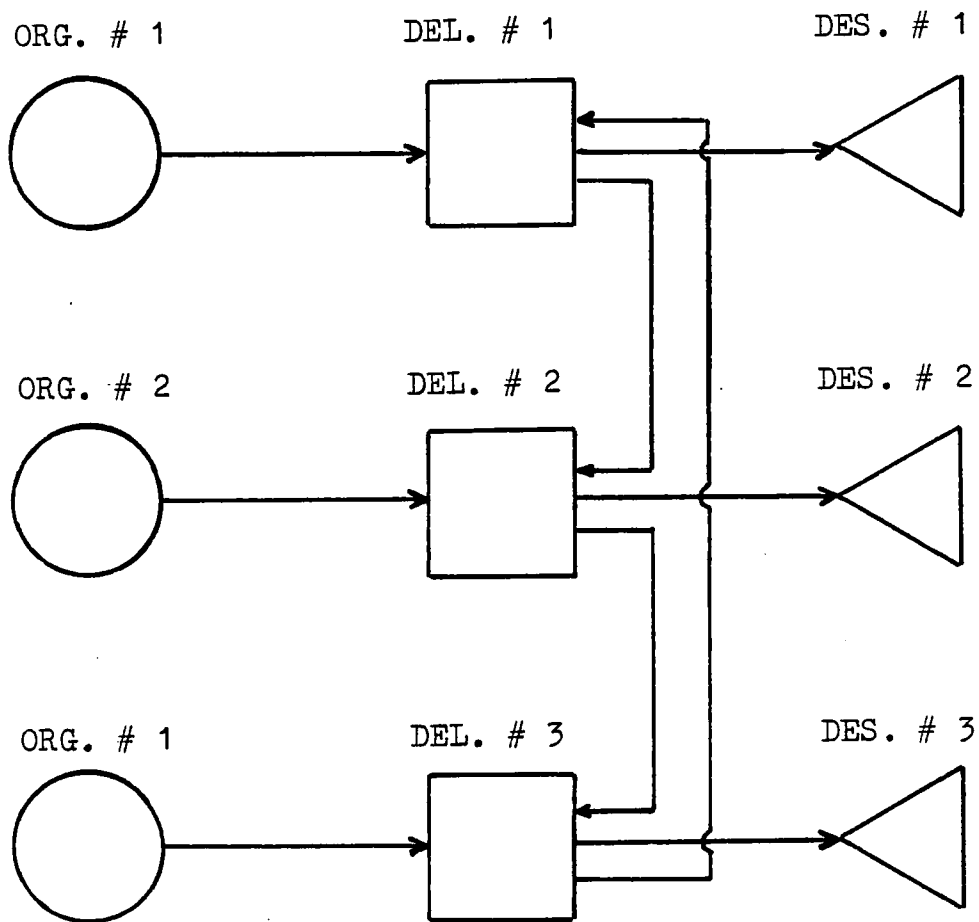


Figure 5.2 Simulation network model

CHAPTER 6

STATIC DESCRIPTION OF THE SIMULATION NETWORK

CHAPTER 6

STATIC DESCRIPTION OF THE SIMULATION NETWORK

A network exists whenever two or more elements interact with one another. A network can also be defined as consisting of nodes and links. Figure 6.1 illustrates a simple network consisting of two nodes and a single link. It shows that the two nodes communicate with one another or alternatively, some information flows between the two nodes via the link.

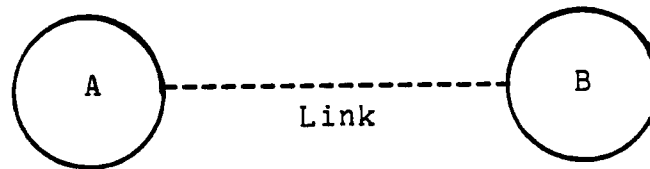


Figure 6.1 A network consisting of two nodes and a link

Figure 6.2 represents a more complex network consisting of four nodes and four links. This illustration shows that each node is a sender and receiver of information. For example, node A is a sender who sends information to node B and C and it also be a receiver who receives information from

node B and C.

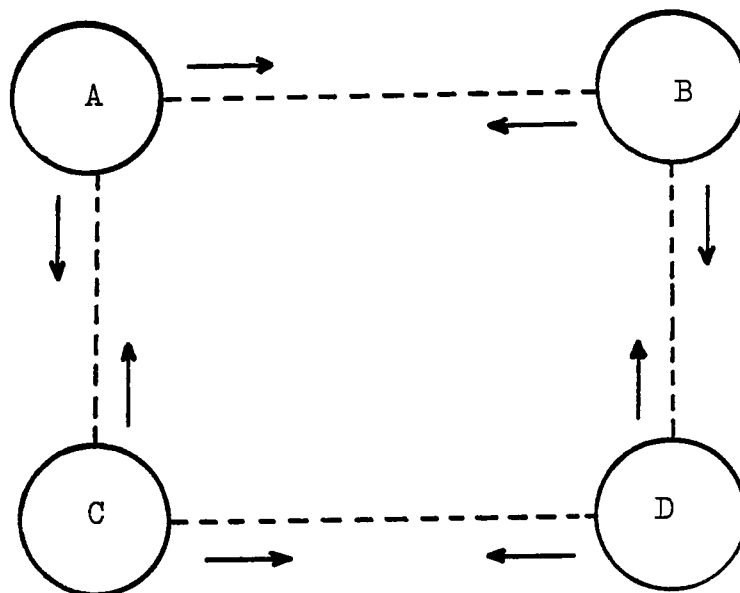


Figure 6.2 A network consisting of four nodes and four links

Our simulation network system is one kind of a network. There are three components of a simulation network :

I. Packets. Packets are the entities that flow through the network. The information of interest about a packet is

1. Time that packet entered system.
2. Transit time of packet in system.
3. Node at which packet entered the system.
4. Node at which packet left system.

II. Nodes. There are three kinds of nodes :

1. Originator node. This node creates packet and sends to another node.
2. Delay node. This node is a single server queue.
3. Destroyer node. This node receives a packet and removes it from the system.

III. Exit path. Exit paths indicate possible flows of packet from node to node.

Detailed description of simulation network components.

The simulation network model consists of an interrelated set of nodes and exit paths. The nodes and exit paths can be considered as elements that are combined and integrated into a system description. Packets are routed along the exit paths emanating from nodes. The exit path taken by a packet is determined randomly. The probability of choice of each exit path emanating from a node is a parameter of that node.

With this brief background, let us proceed to describe the network symbols and definitions of nodes in the system.

An originator node generates packets and routes them into the system over one path that emanates from the originator node. The time for the first packet to be created by each originator node is initialized to be 0.0. A next

creation time for a new packet to be created is specified by the parameters Minwait and Maxwait. The next creation time is current-time + R, where R is a random variable, uniformly distributed on the interval [Minwait,Maxwait].

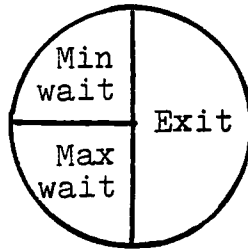
The exit path from the originator node to an other node in the system is specified by the parameter Exit, the number of the target node on the path.

The target of the exit path, Minwait, and Maxwait for each originator node are part of the network model. If we have another network model, we have a different set of these numbers. We use the character '0' to represent an originator node and a positive integer (1,2...,n) to identify each originator node. Figure 6.3 illustrates the symbol for an originator node and summarizes the parameters.

Figure 6.3.a gives an example of an originator node. Originator node number is 1, it has an exit path to delay node number 2 and time until creation of the next packet is in the interval 2.0 - 4.0 .

Figure 6.3.b gives an another example of an originator node. Originator node number is 2, it has an exit path to destroyer node number -5 and time until creation of the next packet is in the interval 3.0 - 5.2 .

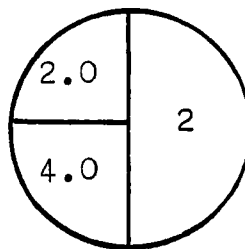
O # 1



Exit : Target node number of exit path.
Minwait : Minimum time until next packet creation.
Maxwait : Maximum time until next packet creation.

Figure 6.3 Symbol and the parameters for each originator node

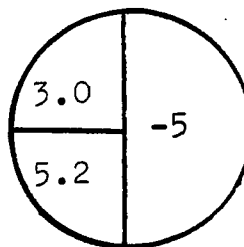
O # 1



Exit = 2
Minwait = 2.0
Maxwait = 4.0

Figure 6.3.a Symbol and the parameters of an originator node number 1

O # 2



Exit = -5
Minwait = 3.0
Maxwait = 5.2

Figure 6.3.b Symbol and the parameters of an originator node number 2

A delay node is used to represent delay in transit from one communication node to another. The delay node is a location in the network where packets wait for service. When a packet reaches a delay node, the service time in that node is computed. If the packet must wait for service, it is stored in the delay queue. The order in which the packets are served from the delay queue for that delay node is specified rank on first-in-first-out (FIFO). Therefore, we have two kinds of delay :

1. Service delay. The computed service time.
2. Delay in queue. The time that a packet waits in the delay queue.

The service time for a packet is S , where S is a random variable, uniformly distributed on the interval [Holdmin, Holdmax].

The parameters Exprob and Exits specify the probability that a packet leaving this node will take a particular exit path. In this implementation, it is possible to specify up to 5 exit paths from each delay node. If less than 5 exits are defined for a particular node, we enter 0 and 0.0 for the number and probability respectively of the unused exit path. To decide which exit path that a packet will take from this delay node we use a random number to choose one of the exit paths from the exit list. More detail of how to compute an

exit path is discussed in Chapter 8.

The targets of the exit paths and their probabilities, Holdmin and Holdmax for each delay node are part of the network model. Different network models have a different value of these numbers. We use the character 'D' to represent a delay node and a positive integer (1,2,...,n) to identify each delay node. Figure 6.4 illustrates the symbol for a delay node and summarizes the parameters.

Figure 6.4.a gives an example of a delay node. The delay node number is 2, it has 5 exit paths to other delay nodes 1, 2, and 3 and to destroyer nodes -3, and -4. The probabilities for exit paths are 0.2,0.3,0.3,0.1,0.1 respectively. The service time in delay node for the packet is in the interval 3.0 - 4.5.

Figure 6.4.b gives an another example of a delay node. The delay node number is 7, it has 3 exit paths to other delay nodes 2, and 5 and to destroyer nodes -10. The probabilities are 0.3, 0.5, and 0.2 respectively. The service time in delay node for the packet is in the interval 2.0 - 2.9.

D # 3

	EX	Ex prob
Holdmin		
Holdmax		

Exits : list of exit paths to other nodes in the system.
 Exprob : probabilities for exit paths of this delay node.
 Holdmin : minimum service time in this delay node.
 Holdmax : maximum service time in this delay node.

Figure 6.4 Symbol and the parameters for each delay node

D # 2

3.0	1	0.2
	2	0.3
4.5	3	0.3
	-3	0.1
	-4	0.1

Exits = 1,2,3,-3,-4
 Exprob = 0.2,0.3,0.3
 0.1,0.1
 Holdmin = 3.0
 Holdmax = 4.5

Figure 6.4.a Symbol and the parameters of delay node number 2

D # 7

2.0	2	0.3
	5	0.5
2.9	-10	0.2
	0	0.0
	0	0.0

Exits = 2,5,-10
 Exprob = 0.3,0.5,0.2
 Holdmin = 2.0
 Holdmax = 2.9

Figure 6.4.b Symbol and the parameters of delay node number 7

A destroyer node represents a node which receives packets and removes them from the network. For this network model, the destroyer node is represented by character 'K' and we use a negative integer (-1,-2,...,-n) to identify each destroyer node. The reason that we use negative integers is to make the destroyer node numbers different from delay node numbers. In this way, exit path targets can be determined from the number alone. Figure 6.5 illustrates the symbol for a destroyer node.

K # -1

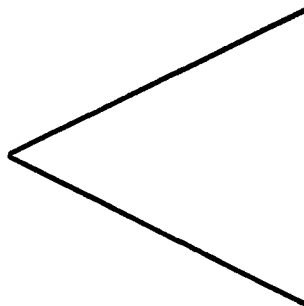


Figure 6.5 Symbol for each destroyer node

To illustrate a simulation network, Table 6.1 gives the parameters for a network with 2 originator nodes, 3 delay nodes and 5 destroyer nodes. Then Figure 6.6 shows the sample of simulation network graphically.

Originator node

Originator node no.	Exit	Minwait	Maxwait
1	1	2.0	4.0
2	2	3.0	5.0

Delay node

Delay node no.	Exits	Exprob	Holdmin	Holdmax
1	2 3 -1 -2 -4	.3 .2 .1 .2 .2	3.0	6.0
2	3 -5 -1 -2 0	.4 .1 .3 .2 .0	2.0	5.0
3	-3 -4 0 0 0	.5 .5 .0 .0 .0	4.0	7.0

Table 6.1 The parameters for originator and delay node in network model

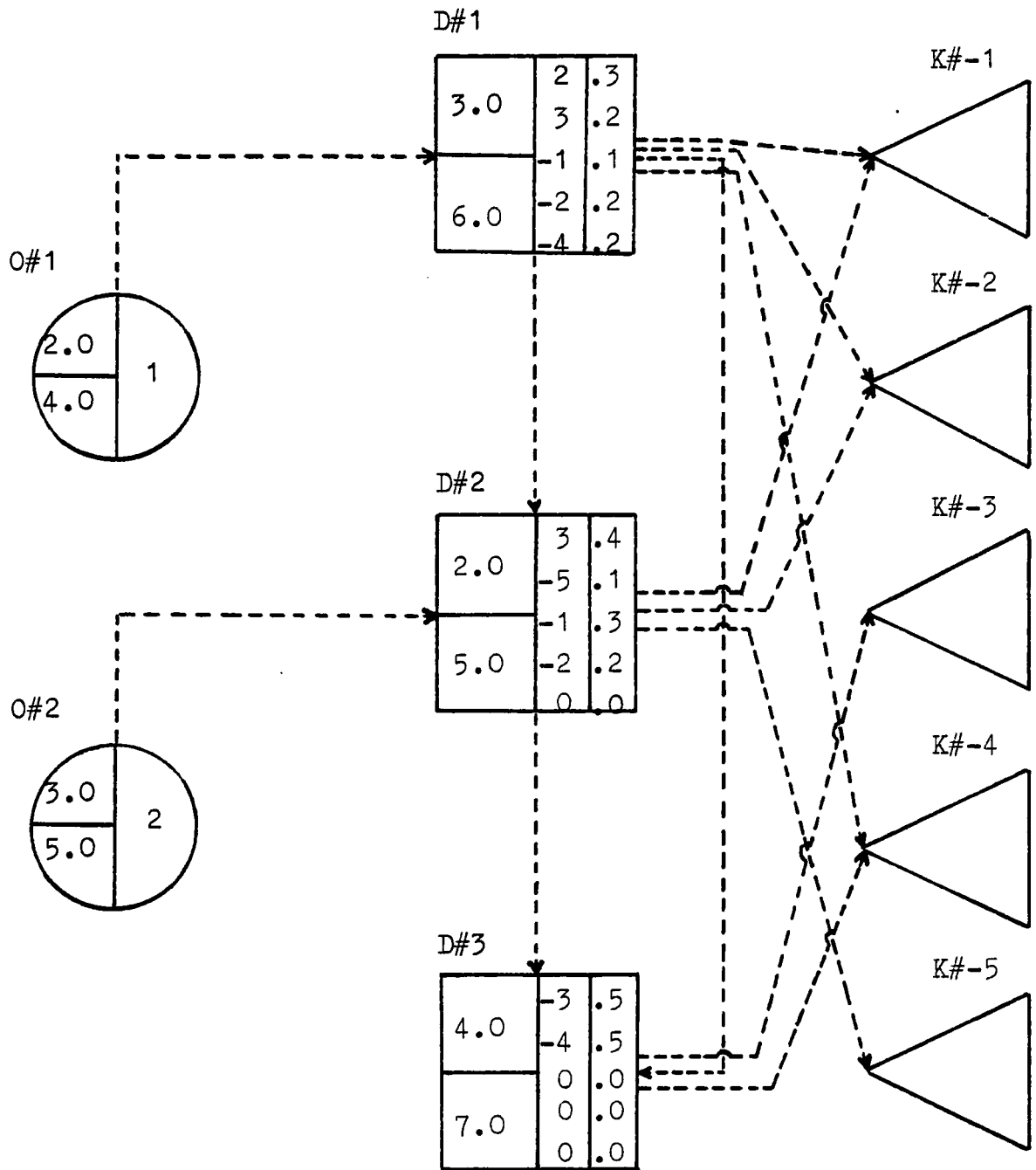


Figure 6.6 Sample of simulation network

CHAPTER 7

DYNAMIC DESCRIPTION OF THE SIMULATION NETWORK

CHAPTER 7

DYNAMIC DESCRIPTION OF THE SIMULATION NETWORK

The activities that occur in our simulation network are shown in Figure 7.1

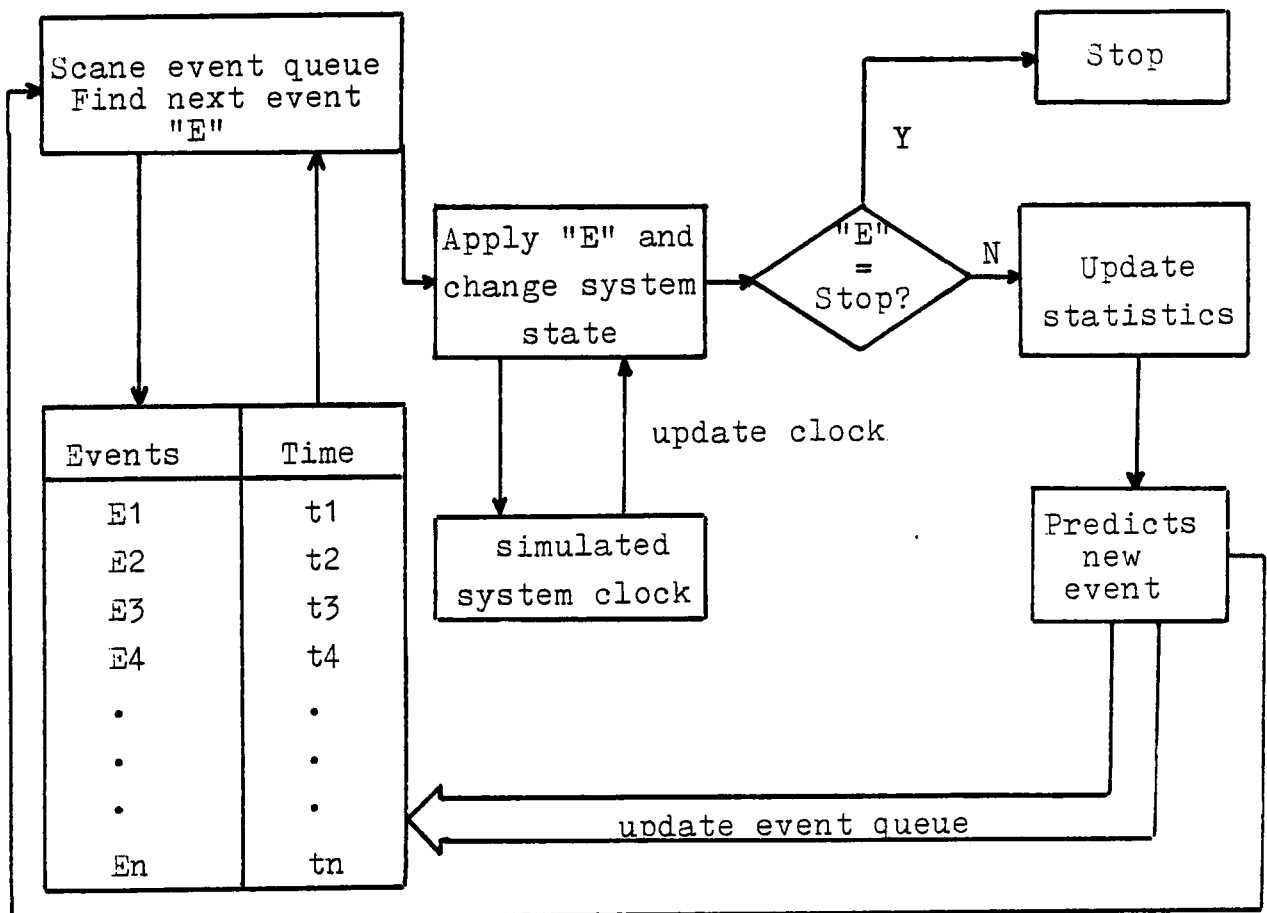


Figure 7.1 Model for Simulation Network Activity

The technique we will use simulation is discrete event simulation. With this method we do not simulate the continuous behavior of a system, only a very specific class of events in the life of the system. We will observe only the events that change the state of the system. The possible types of events that will change the state of our simulation network are

1. Create Event. Create a new packet at an originator node and immediately send it to a delay or a destroyer node.
2. Leave Event. Packet leaves a delay node going to another delay or destroyer node.
3. Stop Event. Stop the simulation program.

The first 2 events that change state in this system can be shown in Figure 7.2

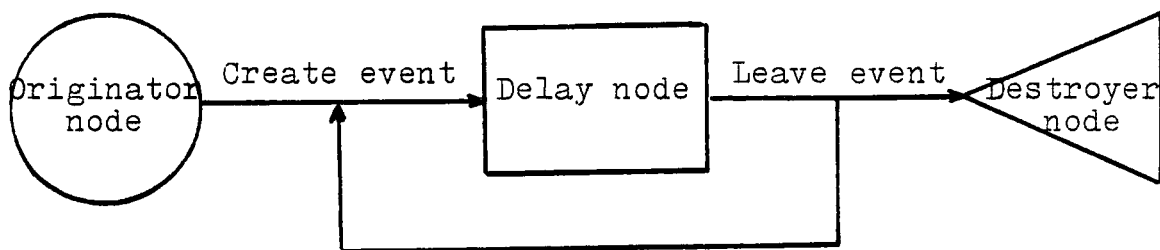


Figure 7.2 Events and State in the System

A data structure, called the event queue, contains a list of all the events to be processed and keeps them sorted by the time they occur. An example of the event queue is shown in Figure 7.3:

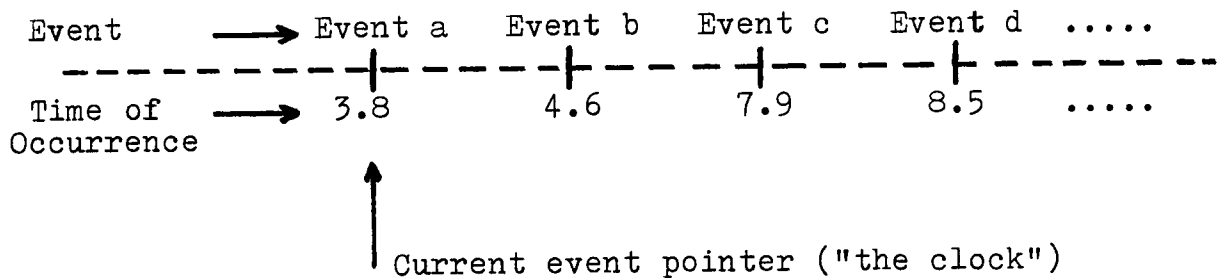


Figure 7.3 List of events in event queue

The highest level of this simulation program can now be explained in terms of the event queue and its events. We move the current event pointer (sometimes called the simulation 'clock') to the next event in the event queue and process that event by changing the system according to the specifications of that event. We continue this until a stop event is encountered. For example, in the preceding diagram, after completing Event a, we would move the current event pointer to time 4.6 and begin processing Event b.

Finally, we must decide how new events will be placed in the event queue. Initially, a stop event is put into the event queue as well as a create event for each originator node. These will be placed there by the procedure Initsys and Initialori. How are other events placed on the event queue? Each event we have defined will create zero or more additional events so that new events are constantly being created. In fact, one of the major purposes of the event procedures will be to schedule new events and place them into the event queue. Thus the simulation will terminate not by running out of events but by reaching the stop event.

There is another detail about simulation that will affect the top level of the program. Simulation is a design tool. This means that we do not know beforehand how to set certain parameters so the system will behave properly. First, we try a set of parameters, note how the system behaves, and adjust the parameters accordingly. Figure 7.4 shows the model of simulation program usage.

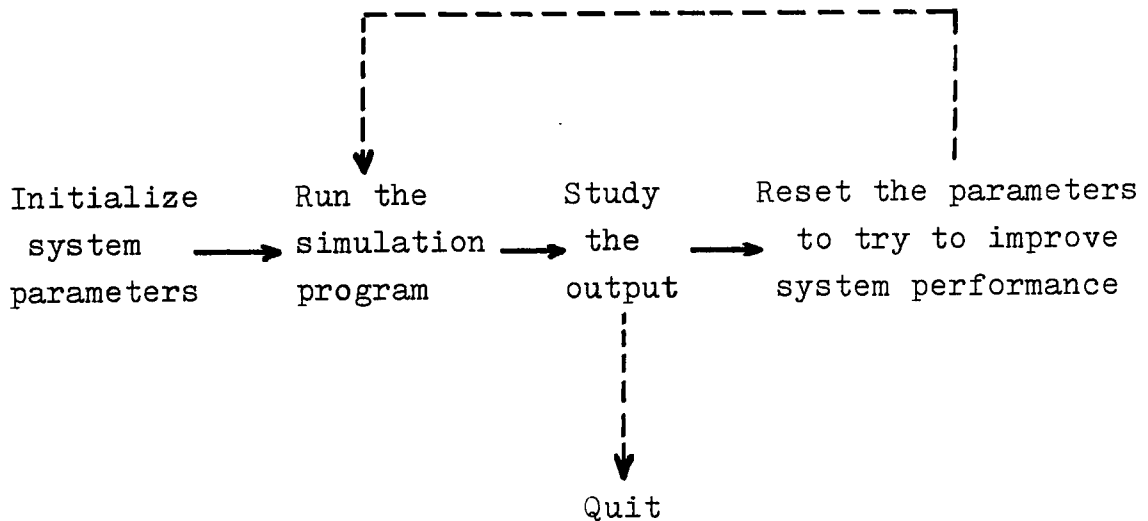


Figure 7.4 Model of Simulation Program Usage

Therefore, the simulation program must be able to run frequently, with a different set of parameters at each time. To this end, The set of parameters will be read from an input data file and we can change those parameters by changing the original file or specifying a different file.

Now let us look at the four event processing routines: Createpacket, Arrivedelaynode, Arrivedestroyernode, and Leavedelay. Before we begin to write these routines, we must make an important decision. Exactly what output results do we want the simulation model to produce? What statistics do we need to understand the behavior of the system? We have chosen the following ten.

1. Number of packets created by each originator node.
2. Total number of packets created by all originator nodes.
3. Number of packets that pass through each delay node.
4. Total number of packets that passed through all delay nodes.
5. Number of packets destroyed in each destroyer node.
6. Total number of packets destroyed by all destroyer nodes.
7. Average delay time for a packet at each delay node.
8. Average delay time in all delay nodes.
9. Average transit time for packets (from creation to destruction) by destroyer node where destroyed.
10. Average transit time for a packet, system wide.

There are fields in each node to collect data in order to compute these statistics

Originator node :

- Count. Counts packets created.

Delay node :

- Mark. Total time that packets have waited in delay node.

- Count. Counts packets in delay node.

Destroyer node:

- Totaltime. Total transit time for all packets.

- Count. Packet counter.

Packet :

- Born. Time that packet was created.
- Wait. Transit time of packet in system.

In this simulation network system, The activities associated with each type of event are

1. Create event (Procedure Createpacket) : The following activities may occur for this event

- (a) Next creation is event scheduled.
- (b) Leave delay node is scheduled.
- (c) Enqueuing in a delay queue.
- (d) Statistics updating.

2. Leave event (Procedure Leavedelay) : The following activities may occur for this event

- (a) Exit target is chosen.
- (b) Leave delay node is scheduled
- (c) Enqueuing in a delay queue.
- (d) Remove packet from system if sent to a destroyer node.
- (e) Statistics updating.

The summary of the event scheduling process is shown in Figure 7.5

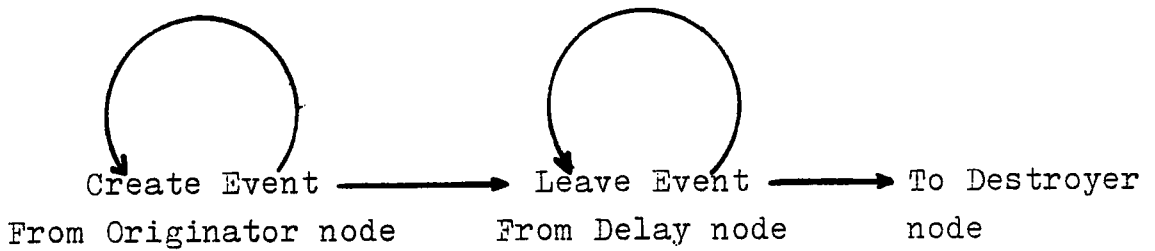


Figure 7.5 Shows the event scheduling process

The procedures dependence for this simulation network is summarized in Figure 7.6

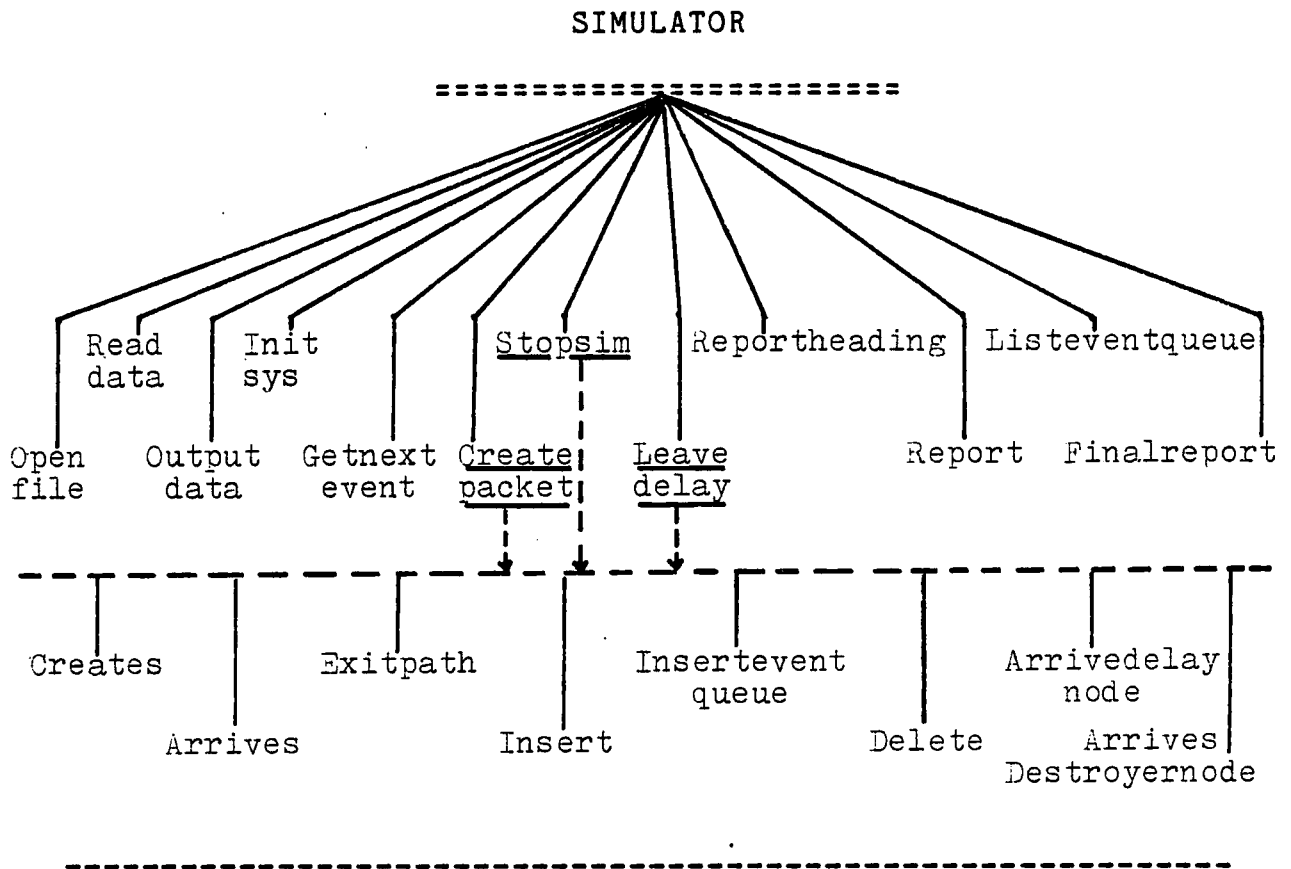


Figure 7.6 Procedures dependence for the simulation network

CHAPTER 8

DOCUMENTATION FOR SIMULATION NETWORK PROGRAM

CHAPTER 8

DOCUMENTATION FOR SIMULATION NETWORK PROGRAM

This program is used to simulate the behavior of a communications network system using discrete event simulation. The program contains 26 procedures and 1 function with approximately 1,200 lines of code (with comments, about 1,561 lines). It exemplifies the top-down design of programs and data structures. This program must be able to be run frequently, with a different set of parameters each time. The first important thing for this program is how to define the data structure for the program.

In this Network System, we have three kind of nodes called Originator, Delay, and Destroyer nodes. We specify the maximum number of each type of node at the beginning of the program. Those numbers are symbolic constants. We can change them if we want to model a larger system. In the present form of the program we allow 3 originator nodes, 7 delay nodes, and 10 destroyer nodes.

We define data structures for each of these parts of the simulated system : clock, event types, event queue,

originator nodes, delay nodes, destroyer nodes and packets. The details for these data structures are described as follow.

Clock :

The simulation network model has a function managing simulation time. The times that are involved in this network model are creation time for new packets and end of service time for a packet in a delay node. These times are what we mean by 'event time'. The clock is the pointer that points to the current event in the event queue and is continually updated during the simulation run. After completing the processing of the first event in the event queue, we move the clock to the next event and process that event by changing the system according to the specifications of that event. The initial value of the clock in our simulation network model is 0.000 .

Event type :

This program is concerned about events that change the state of the system. So, we define an enumeration data type for the possible types of events that can occur. For this system we define

```
Type      Act = (Create,Leave,Stop);
Var       Action : Act;
```

The three kinds of events are as follows

1. Create event. Create a new packet at an originator node and send it to the next node in the system.
2. Leave event. A packet leaves a delay node going to another delay node or a destroyer node.
3. Stop event. This event stops the simulation.

Each event will be represented by a single record containing the following four pieces of information

1. Action. The type of event.
2. Time. Time that the event will occur.
3. Parm. Number of an originator, delay or destroyer node.
4. Pack. Pointer to the packet involved in the event.

This data structure is defined with the following declarations

Type

```
Eventptr = Eventnode
{pointer to an event in eventqueue}
Eventnode =
Record{record of event}
    Action : Act;
    Time    : Real;
    Parm    : Integer;
    Pack    : Ptr;
End; {Record Eventnode}
```

Event queue :

The event queue is a special data structure. It contains a list of all events to be processed and keeps them

sorted by the time they will occur. We use a linked structure to keep a linear list of pending events.

An array is probably an inappropriate data structure for a event queue, since we have no prior knowledge of how many events may be on the event queue at any one time. Therefore a linked list structure is the best way to represent the event queue. By using linked list we have no limit for the maximum number of events in the event queue. The following diagram shows the structure of the event queue.

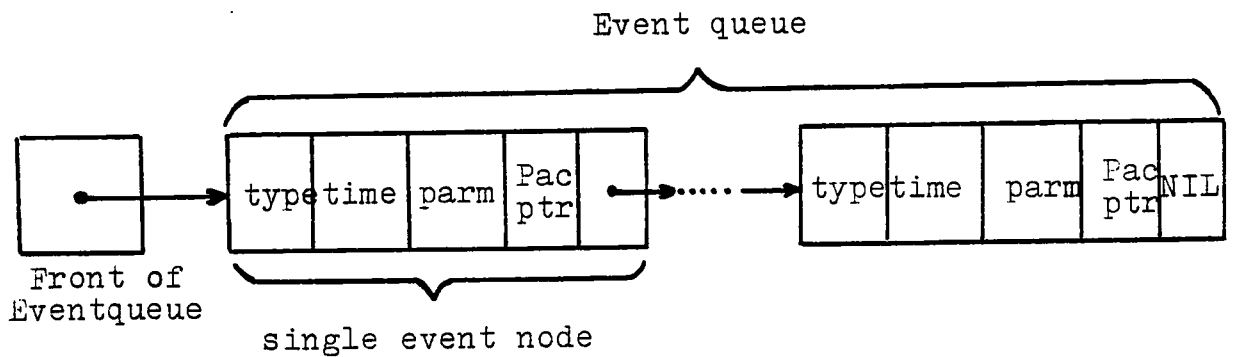


Figure 8.1 The structure of the event queue

The structure is defined through the following declaration

Type

```
Eventqueue =  
Record  
    Efront : Eventptr;  
    {Pointer to the first event in eventqueue}  
    Eback  : Eventptr;  
    {Pointer to the last event in eventqueue}  
    Ecount : Integer; {Counter event in eventqueue}  
End; {End of record eventqueue}
```

In order to avoid having to pass parameters from procedure to procedure we will define 3 important record structures named Originator, Delay and Destroyer to represent originator nodes, delay nodes and destroyer nodes respectively. The definition for each record will look like this

Delay node :

Type

```
Delay =  
Record  
    Q          : Queue;  
    {Pointer to wait queue for this delay node}  
    Exits      : Array [1..Noexits] of Integer;  
    {Exits list of target node}  
    Exprob     : Array [1..Noexits] of Real;  
    {Exits probability of target node}  
    Holdmin    : Real;  
    {Minimum service time in delay node}  
    Holdmax    : Real;  
    {Maximum service time in delay node}  
    Mark       : Real;{Total service time in delay node}  
    Aver       : Real;{Avg.service time in delay node}  
    Waited     : Real;  
    {Service time for a packet in delay node}  
    Wait       : Real;{Waiting time in delay queue}  
    Last       : Real;  
    {Time that last packet leave delay node}  
    Count      : Integer;{Counter packet in delay node}  
    Calendar   : Eventptr;{Pointer event in event queue}  
End; { Record Delay }
```

We name each delay node by positive integer and we define an array [1..N] of individual delay node records.

Each delay node has a queue for holding packets waiting for service called the delay queue. This is a FIFO queue. We implement a delay queue as a linked list of individual packet records. We cannot use an array to implement a delay queue because we have no idea what the maximum number of packets in a queue will be at any one time. For example if we declare our array to contain 100 packets for each delay node, what would happen if there were 101 packets waiting in the delay node? The following diagram shows the structure of a delay queue.

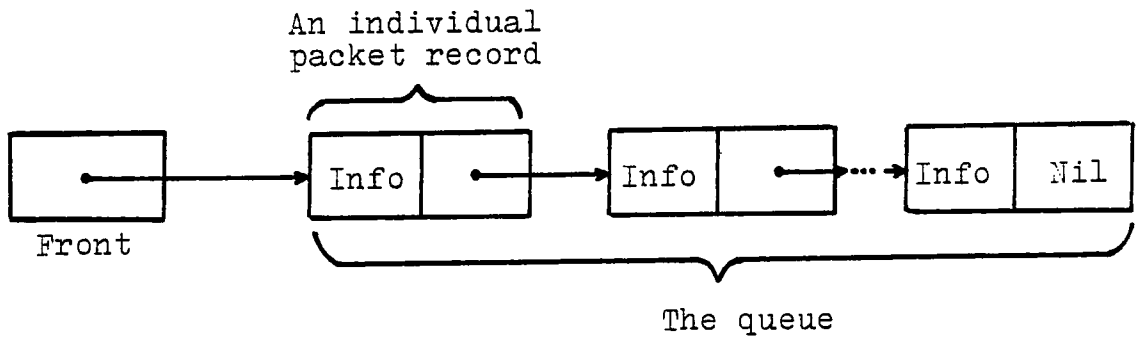


Figure 8.2 The structure of a delay queue

The structure is defined through the following declaration.

Type

```
Queue =  
Record  
    Front : Ptr;  
    {Pointer to the first packet in delay queue}  
    Back  : Ptr;  
    {Pointer to the last packet in delay queue}  
    count : Integer; {Counter packet in delay queue}  
End; {End of record queue}
```

Originator node :

Type

```
Originator =  
Record  
    Exit      : Integer; {Target node from Ori.}  
    Minwait   : Real;  
    {Minimum time until next packet creation}  
    Maxwait   : Real;  
    {Maximum time until next packet creation}  
    Count     : Integer; {Counter packet created}  
    Curtime   : Real; {Next packet creation time}  
    Calendar  : Eventptr;  
    {Pointer to the create event in the event  
    queue}  
End; {Record originator}
```

We name each originator node by positive integer and we define an array [1..N] of individual originator records.

Destroyer node :

Type

```
Destroyer =  
Record  
    Transit   : Real;  
    {Transit time for most recent packet}  
    Totaltime : Real;  
    {Total time for all packets since created  
    until destroyed}  
    Count     : Integer;  
    {Counter of packets destroyed}  
End; {Record Destroyer}
```

We use negative integer name the destroyer nodes. This allows us to distinguish delay nodes and destroyer nodes by name when specified as the destination for a packet. So we define an array [-N..-1] of individual destroyer records.

The goal of this program is to collect information concerning the performance of the simulated network. Therefore, the waiting time in each node and the total time for transit of the packet from creation to destruction are very important. We incorporate these value as fields in the record structure for the packet. We also need to keep track of certain information for debugging purposes.

Packet :

Packets are the entities that flow through the network. Each packet will be represented by a single record containing the following five pieces of information :

1. Born. Time that packet was created.
2. Marks. Waiting time of packet in each delay node.
3. Wait. Transit time of packet since create until destroy (used in debugging report).
4. Org. Originator node that created the packet.
5. Nex. Time that next packet leave from delay queue (used in debugging report).
6. Pacno. Serial number of this packet relative to its originator node. Org and Pacno uniquely identify each packet.

These data structures is defined with the following declarations.

Type

```
Ptr = Packet;
Packet =
Record
    Born : Real;
    Marks : Real;
    Wait : Real;
    Org : Integer;
    Nex : Real;
    Pacno : Integer;
End; {Record Packet}
```

In the preceding paragraph, we talked about three event types in the system, they are Create events, Leave events, and the Stop event. The highest level of this simulation program can be explained in terms of the event queue and its events. Therefore, the main loop of this program and the routines involved in each kind of event can be described as follows:

```
Repeat
    Get next event from eventqueue;
    Update clock;
    Perform action;
    If debug requested then
        Report after performed action;
Until action = stop
```

Create event :

This event creates a new packet at an originator node and send it to the next node in the system. The loop of this event can be described as follows:

```

Begin
    Create a packet;
    Compute next creation time;
    Schedule next creation time in the event
    queue;
    Send packet to a delay or destroyer node;
    Compute service time or transit time of
    a packet;
    Schedule service time in the event queue;
End;

```

The routines involved in a Create event are

1. Creates. This routine computes the next creation time for new packet.
2. Insert. This routine puts a packet in a delay queue of a delay node.
3. Arrives. This routine computes service time of a packet at a delay node.
4. Arrivedelaynode. This routine sends a packet to a delay node.
5. Arrivedestroyernode. This routine sends a packet to a destroyer node.
6. Inserteventqueue. This routine inserts a new event, keeping order by time, into the event queue.

Leave event :

This event sends a packet from delay node to another delay node or a destroyer node. The loop of this event can be described as follows:

```

Begin
    Compute exit path from delay node;
    Remove packet from delay node;
    If packet is sent to delay node then
    Call procedure Arrivedelaynode;
    Else
    Call procedure Arrivedestroyernode;
End;

```

The routines involved in a leave event are

1. Exitpath. This routine computes exit from a delay node.
2. Delete. This routine removes a packet from a delay queue of a delay node.
3. Arrives. This routine computes service time of a packet at a delay node.
4. Arrivedelaynode. This routine sends a packet to another delay node.
5. Arrivedestroyernode. This routine sends a packet to a destroyer node.
6. Inserteventqueue. This routine inserts a new event, keeping order by time, into the event queue.

Stop event :

The routine involved in Stop event is

1. Stopsim. Routine stop simulation program.

Arrivedelaynode :

This procedure sends a packet to a delay node. The loop of this procedure can be described as follows:

```
Begin
  If delay queue is empty then
    Compute service time for a packet;
    Schedule into the event queue;
  Else
    Compute waiting time in delay queue;
    Schedule into the event queue;
    Schedule another leave event into
    the event queue;
End;
```

Arrivedestroyernode :

This procedure sends a packet to a destroyer node. The loop of this procedure can be described as follows:

```
Begin
    Compute transit time for a packet;
    Update total transit time for all
    packets;
End;
```

Those routines are some parts of simulation program. We are now in a position to explain the top level of the simulation program. This level consists of eleven procedures as follows:

1. Openfile
2. Readdata
3. Outputdata
4. Initsys
5. Getnextevent
6. Createpacket
7. Leavedelay
8. Reportheading
9. Report
10. Listeventqueue
11. Finalreport

Procedure Openfile

This procedure opens the input data file and output data file. This program will accept different input data files. The user can have one, or more than one, input data file with different values of data. The user specifies the input data file interactively at the beginning of the program. The user can also specify an output data file interactively. For example, if we have two input data files called A.DAT and AA.DAT we can select one of those files to run the program and for the results of the program we can name the output file called B.DAT or something else. The user may choose how to name the output file.

Procedure Readdata

This procedure reads data for originator and delay nodes from the input data file. The format of input data file is described in Chapter 9. The data for each originator node are three fields of an Originator record : exit, minwait, and maxwait. The data for each delay node are four fields of a Delay record : exits, exprob, holdmin, and holdmax. From the input data file we also read the number of originator nodes, number of delay nodes, and number of exits from a delay node.

Procedure Outputdata

This procedure prints statistical summaries for a

simulation run.

Procedure Initsys

This procedure initializes the system simulation, and construct data tables for originator and delay nodes. It also requests an initial value of the seed in order to generate random numbers, and requests an option for report after performing action of the event. This procedure also initializes summary values for all originator, delay, and destroyer nodes.

Procedure Getnextevent

This procedure gets the next event from event queue, removing it from the front of the event queue.

Procedure Createpacket

This procedure is performed for a Create event. It allocates storage for a new packet. It also initializes fields in the new packet then, it schedules a creation time for another packet. After the packet has been created this procedure routes it to a delay or destroyer node immediately depending on the exit parameter of the originator node.

If the packet goes to a delay node and the delay queue is empty, determine the leaving time from this delay node for that packet and schedule a Leave event. If delay queue is non-empty compute the waiting time that packet wait in delay

queue, then determine the leaving time for that packet and also schedule a Leave event into the event queue.

If the packet goes to a destroyer node, compute the transit time for that packet and update total transit time for this destroyer node.

Procedure Leavedelay

This procedure removes a packet from a delay node routing it to another delay node or to a destroyer node. Another procedure, Exitpath computes the exit path for a delay node. If the exit number is positive, it means send the current packet to another delay node. In this case, procedure Arrivedelaynode is called. If the exit number is negative, it means send the current packet to a destroyer node. In this case, procedure Arrivedestroyernode is called.

Procedure Reportheading

This procedure constructs the heading for the debug report.

Procedure Report

This procedure reports after the occurrence of an event.

Procedure Listeventqueue

This procedure lists all of the events in the event queue.

Procedure Finalreport

This procedure produces the summary report of the simulation network at the end of a simulation run.

At the next level of detail, the procedures Createpacket and Leavedelay define a total of eight second-level modules. These modules fall into four classes as follows :

(a) Procedures that produce random choices.

- Creates. Compute the creation time for next packet.
- Arrives. Compute the service time for a packet in delay node.
- Exitpath. Compute the exit path from a delay node.

(b) Procedures that manage the delay queue.

- Insert. Adds the new packet to the end of the delay queue.
- Delete. Takes the first packet from the front of the delay queue.

(c) Procedures that send the packet to delay or destroyer node.

- Arrivedelaynode.
- Arrivedestroynode.

(d) Procedure that manages the event queue.

- Inserteventqueue. Adds a new event to the event queue.

We give more details about some procedures as follows :

(1) - Procedure Creates

This procedure computes the creation time of the next packet at an originator node. This creation time is chosen randomly from the interval [Minwait,Maxwait] with uniform distribution.

(2) - Procedure Arrives

This procedure computes the service time for a packet in a delay node. This time value is chosen randomly from the interval [Holdmin,Holdmax] with uniform distribution.

(3) - Procedure Exitpath

This procedure determines the exit path from a delay node. The exit path is chosen randomly from an array of Exits [i], the probabilities of choosing Exits [i] is a array of Exitprob [j]. This is implemented as follows :

A pseudo random number U , uniform from $[0,1]$, is generated. Then subtract each Exitprob [j] from U one at a time. If the value after subtraction is greater than zero, continue by using next Exitprob [j] until the value is less than or equal to zero. The corresponding Exit [i] is the

chosen exit path.

The three routines Creates, Arrives and Exitpath use a dependent routine called random(x,y). This function will generate a random real number, r, in the range $x \leq r \leq y$.

(4) - Procedure Arrivedelaynode

This procedure inserts a packet at the end of a delay queue and counts number of packets in a delay node. If delay queue is empty, it computes the service time and leaving time for a packet then, schedule a Leave event for that packet. If delay queue is not empty, this procedure computes the service time and determines the leaving time for that packet and also updates the waiting time in a delay node. Then it schedules a Leave event for that packet.

(5) - Procedure Arrivedestroynode

This procedure computes the transit time for a packet and updates the total transit time in a destroyer node. It also counts the number of packets that are destroyed in this destroyer node. Then it removes a packet from the system.

(6) - Procedure Inserteventqueue

This procedure inserts a new event into the event queue. The new event is placed according to the time of that event, so that the event list is ordered by the time field.

Finally, we describe the body of this program, sometimes called the Main Program. The main program for our simulation network can be broken down into four parts as follows :

1. Read data for simulation run. This part requests the maximum time for the simulation run and also reads the data from the input data file.
2. Initialize the simulation system. This part initializes values for all originator nodes, delay nodes, and destroyer nodes.
3. Main loop to perform the actions of each event and also update the simulation clock.
4. Print results of the simulation run.

The details of coding for the main program are shown in Table 8.1

```

Begin (* beginning simulation program *)
  Writeln('Welcome to the simulation system for communication
          network');
  Repeat
    Repeat
      Writeln('Please enter maxtime..the total
              simulation run time');
      Readln(Maxtime);
      If maxtime > 0.0 then
        Gooddata := True
      Else
        Begin
          Gooddata := False;
          Writeln('You made an error in the input,);
          Writeln('Maxtime must be real and
                  non-negative number');
          Writeln('Please try again');
        End;
    Until Gooddata;
  Openfile;
  Readdata>(*Read data from input data file*)
  Initsys>(*Initialize the system simulation*)
  Repeat
    If eveq.efront.time <= maxtime then
      (*simulation clock <= maxtime*)
      d := eveq.efront;
      eveq.efront.time := d.time
      Case action of
        Create : Createpacket;
        Leave  : Leavedelay;
        Stop   : Stopsim;
      End; (* of case *)
      Report; (*Report after perform action*)
      Listeventqueue; (*show events in event queue*)
    Until action = stop;
  Listeventqueue;
  Finalreport>(*print final results*)
  Writeln('Do you wish to run the
          simulation network again..');
  Writeln('for a different time value, or
          different input data file?');
  Write('Type Y<es>, or N<o> : ');
  Readln(Select);
  Done := (Select = 'n');
  Until Done;
  Writeln('Exit from the simulation program, thank you');
End. (* End of simulation program *)

```

Table 8.1 Coding Main Program List of Simulation Network

This completes the documentation for the simulation network program. Figure 8.3 gives flowcharts for the components.

A complete listing code of the simulation network program appears on page 60 - 84.

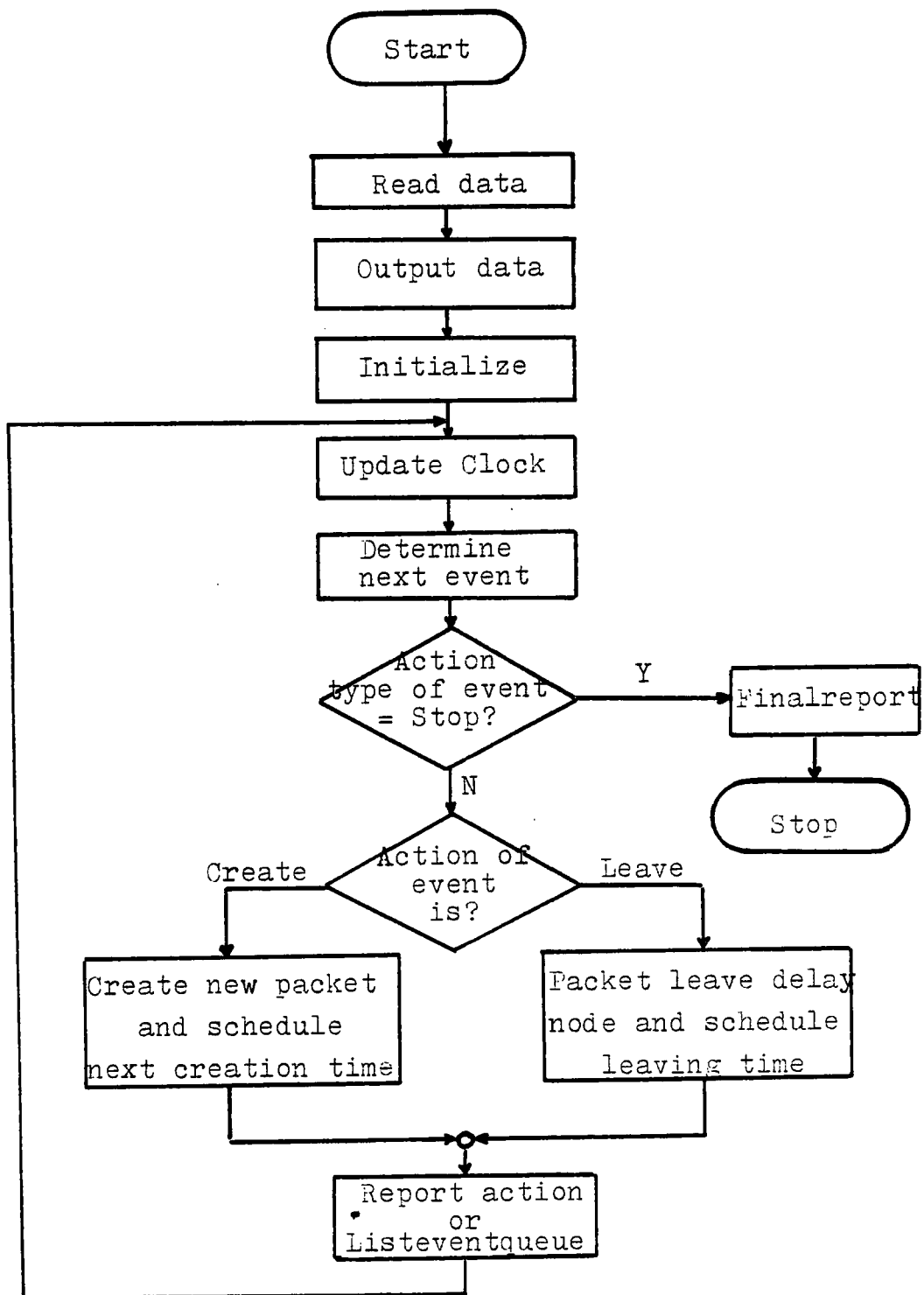


Figure 8.3 Flow Chart for the Simulation Network Program

```
PROGRAM SIMULATION(INPUT,OUTPUT,INFILE,OUTFILE);
C SIMULATION - SIMULATE COMMUNICATION NETWORK SYSTEM; AUG.15,1983
```

```
THIS IS A PROGRAM TO SIMULATE THE BEHAVIOR OF A COMMUNICATION NETWORK SYSTEM
USING DISCRETE EVENT SIMULATION. THE SYSTEM IS DESCRIBED IN CHAPTER 3 OF
THIS THESIS THAT SUBMIT TO MATH. AND COMPUTER SCIENCE DEPARTMENT, ATLANTA
UNIVERSITY, FOR A GRADUATE STUDENT; BY TAKSIN PLABJANG. }
```

```
CONST
```

```
NOEXITS = 5; (* NUMBER EXITS OF DELAY NODE *)
ORIGS = 3; (* NUMBER OF ORIGINATOR NODES *)
DELAS = 7; (* NUMBER OF DELAY NODES *)
DESTS = -10; (* NUMBER OF DESTROYER NODES *)
STARTTIME = 0.00; (* SIMULATE START TIME *)
```

```
TYPE
```

```
ACT = (CREATE,LEAVE,STOP);
(* POSSIBLE TYPES OF EVENTS THAT CAN OCCUR *)
```

```
PTR = ^PACKET;
```

```
PACKET =
```

```
RECORD
```

```
BORN : REAL; (* TIME THAT PACKET CREATED *)
ORG : INTEGER; (* ORIGINATOR NODE THAT CREATE PACKET *)
MARKS : REAL; (* WAITING TIME IN EACH DELAY NODE *)
WAIT : REAL; (* TOTAL TIME THAT WAIT IN DELAY NODES *)
NEX : REAL; (* TIME THAT NEXT PACKET LEAVE DELAY QUEUE *)
PACNO : INTEGER; (* NUMBER OF PACKET HAD BEEN CREATED *)
NEXT : PTR;
```

```
END; (*RECORD PACKET*)
```

```
EVENTPTR = ^EVENTNODE;
```

```
EVENTNODE =
```

```
RECORD
```

```
ACTION : ACT;
TIME : REAL; (* TIME OF OCCURRENCE *)
PARG : INTEGER; (*NUMBER OF ORIGINATOR,DELAY,OR DESTROYER NODE*)
PACK : PTR;
LINK : EVENTPTR;
```

```
END; (*RECORD EVENTNODE*)
```

```
QUEUE = (*DATA STRUCTURE FOR KEEP RECORD OF PACKET*)
RECORD
```

```
FRONT : PTR;
BACK : PTR;
COUNT : INTEGER; (*COUNTER PACKET IN QUEUE*)
```

```
END; (*RECORD QUEUE*)
```

```
EVENTQUEUE = (*DATA STRUCTURE FOR KEEP EVENTS OF THE SYSTEM*)
RECORD
```

```
EFRONT : EVENTPTR;
EBACK : EVENTPTR;
ECOUNT : INTEGER; (*COUNTER EVENT IN EVENTQUEUE*)
```

```
END; (*RECORD EVENTQUEUE*)
```

```
ORIGINATOR =
```

```
RECORD
```

```
EXIT : INTEGER; (* EXIT TO DELAY NODE *)
MINWAIT : REAL; (* MINIMUM TIME UNTIL NEXT PACKET CREATION *)
MAXWAIT : REAL; (* MAXIMUM TIME UNTIL NEXT PACKET CREATION *)
COUNT : INTEGER; (* COUNTER PACKET CREATED *)
CURTIME : REAL; (* TIME THAT CREATE PACKET *)
CALENDAR : EVENTPTR;
```

```
END; (*RECORD ORIGINATOR NODE*)
```

```
ORIG = ARRAY[1..ORIGS] OF ORIGINATOR;  
      (*DATA STRUCTURE FOR KEEP ORIGINATOR NODES*)
```

```
DELAY =  
RECORD  
  Q      : QUEUE;  
  EXITS  : ARRAY[1..NOEXITS] OF INTEGER;  
          (*EXITS LIST OF DELAY NODE*)  
  EXPROB : ARRAY[1..NOEXITS] OF REAL;  
          (*EXITS PROBABILITY OF DELAY NODE*)  
  HOLDMIN : REAL; (* MINIMUM SERVICE TIME IN DELAY NODE *)  
  HOLDMAX : REAL; (* MAXIMUM SERVICE TIME IN DELAY NODE *)  
  MARK    : REAL; (* TOTAL TIME THAT WAIT IN DELAY NODE *)  
  AVER    : REAL; (* AVERAGE TIME IN DELAY NODE *)  
  WAITED  : REAL; (* WAITING TIME FOR A PACKET IN DELAY NODE  
                  BEFORE LEAVING *)  
  WAIT    : REAL; (* WAITING TIME IN DELAY QUEUE *)  
  LAST    : REAL; (* TIME THAT LAST PACKET LEAVE DELAY NODE *)  
  COUNT   : INTEGER; (* COUNTER PACKET IN DELAY NODE *)  
  CALENDAR : EVENTPTR;  
END; (*RECORD DELAY NODE*)
```

```
DELA = ARRAY[1..DELAS] OF DELAY;  
      (*DATA STRUCTURE FOR KEEP DELAY NODE*)
```

```
DESTROYER =  
RECORD  
  TRANSIT : REAL; (*TRANSIT TIME FOR EACH PACKET*)  
  TOTALTIME : REAL;(*TOTALTIME FOR ALL PACKETS SINCE CREATED  
                   UNTIL DESTROYED*)  
  COUNT    : INTEGER; (* COUNTER PACKET IN DESTROYER NODE *)  
END; (*RECORD DESTROYER NODE*)
```

```
DEST = ARRAY[DESTS..-1] OF DESTROYER;  
      (* DATA STRUCTURE FOR KEEP DESTROYER NODE *)
```

```
NAMED = PACKED ARRAY[1..10] OF CHAR;  
      (* TO IDENTIFIED NAME OF INPUT AND OUTPUT FILE *)
```

VAR

```
(* GLOBAL VARIABLES *)  
  
PARM,NUMBER : INTEGER;  
(*SPECIFY NUMBER OF ORIGINATOR,DELAY OR DESTROYER NODE IN EVENTQUEUE*)  
TIME : REAL; (* TIME OF NEXT EVENT IN EVENTQUEUE *)  
ACTION : ACT; (* TYPE OF NEXT EVENT IN EVENTQUEUE *)  
CALENDAR : EVENTPTR; (* EVENT INVOLVED IN CURRENT EVENTQUEUE *)  
ORI : ORIG; (* ARRAY OF ORIGINATOR NODE RECORD USE FOR INITIAL  
            VALUE AND POINTER TO ORIGINATOR NODE *)  
DEL : DELA; (* ARRAY OF DELAY NODE RECORD USE FOR INITIAL VALUE  
            AND POINTER TO DELAY NODE *)  
DES : DEST; (* ARRAY OF DESTROYER NODE RECORD USE FOR INITIAL  
            VALUE AND POINTER TO DESTROYER NODE *)  
EVTIME : REAL; (* TIME THAT EVENT IS TO BE SCHEDULED IN EVENTQUEUE *)  
EVEQ : EVENTQUEUE; (* POINTER TO THE EVENT IN EVENTQUEUE *)  
A : ACT; (* TYPE OF THE EVENT IN EVENTQUEUE *)  
NPTR : PTR; (* PACKET INVOLVED IN CURRENT EVENT *)  
RANSEED : INTEGER;  
(* INITIAL VALUE OF SEED NUMBER FOR GENERATE RANDOM NUMBER *)  
NO,XX,YY : INTEGER; (* INITIALIZE VALUE OF NODES TYPE IN THE SYSTEM *)  
DELNO,ORIND,DELNOS,DESNO,EXNO,NOS : INTEGER;  
(* SPECIFY NUMBER OF NODES TYPE IN THE CURRENT EVENT *)
```

```

NODES,NODED : PACKED ARRAY(1..10) OF CHAR;
(* SPECIFY NODE TYPE IN THE SYSTEM *)
INF,OUTF : NAMED; (* TO IDENTIFY NAME OF INPUT AND OUTPUT FILE *)
INFILE,OUTFILE : TEXT;
(* OPEN INFILE FOR READ STATISTICAL DATA, AND OUTFILE FOR RESULTS OF
THIS SIMULATION PROGRAM *)

```

```

(* MAIN BLOCK LOCAL VARIABLES *)

```

```

DONE : BOOLEAN; (* BOOLEAN FLAG TO SIGNAL COMPLETION *)
NO3 : INTEGER; (* NUMBER OF EXIT FROM ORIGINATOR NODE *)
NO2 : INTEGER; (* NUMBER OF EXIT FROM DELAY NODE *)
TIM : REAL; (* NEXT CREATION TIME FOR NEW PACKET *)
ORIS : ORIG;
(* ARRAY FOR ORIGINATOR NODE RECORD USE FOR SPECIFY ORIGINATOR NODE
THAT INVOLVED IN CURRENT EVENT *)
DELS : DELA;
(* ARRAY FOR DELAY NODE RECORD USE FOR SPECIFY DELAY NODE THAT INVOLVED
IN CURRENT EVENT *)
DESS : DEST;
(* ARRAY FOR DESTROYER NODE RECORD USE FOR SPECIFY DESTROYER NODE
THAT INVOLVED IN CURRENT EVENT *)
MAXT : REAL; (* MAXIMUM TIME FOR SIMULATION RUN *)
TT : REAL; (* TIME OF OCCURENCE EVENT *)
TEM4 : REAL; (* LEAVING TIME OF THE PACKET FROM DELAY NODE IN
CASE DELAY QUEUE IS EMPTY *)
TEM6 : REAL; (* LEAVING TIME OF THE PACKET FROM DELAY NODE IN
CASE DELAY QUEUE IS NOT EMPTY *)
TEM2 : REAL; (* LEAVING TIME OF THE PACKET FROM ANOTHER DELAY
NODE AFTER COMPUTE EXITPATH, QUEUE EMPTY *)
TEM3 : REAL; (* LEAVING TIME FROM ANOTHER DELAY NODE, IN CASE
QUEUE IS NOT EMPTY *)
NPVAL : EVENTPTR; (* POINTER TO THE EVENT THAT TO BE DELETED FROM
EVENTQUEUE *)
D : EVENTPTR; (* POINTER TO THE FIRST EVENT IN EVENTQUEUE *)
PACNO : INTEGER; (* INITIAL VALUE OF NUMBER PACKET IN THE SYSTEM *)
SELECT,SELECT1,SELECT2 : CHAR; (* USE FOR SELECT ABOUT CONTINUATION *)
GOODDATA : BOOLEAN; (* BOOLEAN FLAG USED TO VALIDATE INPUT *)

```

```

C OPENFILE:

```

```

THIS PROCEDURE OPEN AND RESET DATA FILE FOR STATISTIC DATA OF A COMMUNICA-
TION NETWORK SYSTEM AND ALSO OPEN AND REWRITE OUTPUT FILE FOR THE RESULTS
OF THIS SIMULATION PROGRAM.

```

```

PROCEDURE OPENFILE;

```

```

BEGIN
WRITELN('PLEASE TYPE NAME OF INPUT DATA FILE ....');
READLN(INF);
WRITELN(INF);
OPEN(INFILE,INF,HISTORY := OLD);
RESET(INFILE);
WRITELN;
WRITELN('PLEASE TYPE NAME OF OUTPUT DATA FILE ...');
READLN(OUTF);
WRITELN(OUTF);
OPEN(OUTFILE,OUTF,HISTORY := NEW);
REWRITE(OUTFILE);
WRITELN;
END; (*OPEN FILE*)

```

```

C UNDERLINE:

```

THIS PROCEDURE PRINT THE DASHED-LINE USE FOR FORMATTING THE OUTPUT OF THIS PROGRAM. }

```
PROCEDURE UNDERLINE;
CONST WID = 80;
VAR
    II : INTEGER;
BEGIN
    FOR II := 1 TO WID DO
        BEGIN
            WRITE('-');
        END;
    END; (*UNDERLINE*)
```

{ HEADING:
THIS PROCEDURE PRINT THE STARS USED FOR FORMATTING THE OUTPUT OF THIS PROGRAM. }

```
PROCEDURE HEADING;
CONST WIDTH = 80;
VAR
    I : INTEGER;
BEGIN
    FOR I := 1 TO WIDTH DO
        BEGIN
            WRITE('*');
        END;
    END; (*HEADING*)
```

{ READDATA:
THIS PROCEDURE READ DATA FROM DATA FILE THAT CONTAIN THE DATA OF 3 ORIGINATOR NODES AND 7 DELAY NODES.

INPUT PARAMETERS :

- ORI : 3 FIELDS OF RECORD TYPE NAMED ORIGINATOR [EXIT,MINWAIT AND MAXWAIT]
- DEL : 4 FIELDS OF RECORD TYPE NAMED DELAY [EXITS,EXPROB,HOLDMIN, HOLDMAX]

OUTPUT PARAMETERS :

- ORI : USE THOSE VALUES TO COMPUTE AND DETERMINE THE ULTIMATE OVERALL ACCURACY OF THE SYSTEM.
- DEL : USE THOSE VALUES TO COMPUTE AND DETERMINE THE ULTIMATE OVERALL ACCURACY OF THE SYSTEM. }

```
PROCEDURE READDATA;
VAR
```

```
    INX : INTEGER; (* COUNTER FOR ORIGINATOR NODES *)
    INDX : INTEGER; (* COUNTER FOR NUMBER OF EXITPATH *)
    ORIGSS, DELASS, NOEXITSS : INTEGER;
    (* NUMBER OF ORIGINATOR, DELAY NODE AND EXITPATH *)
```

```
BEGIN
    WRITELN('READING THE DATA FOR SYSTEM SIMULATION NETWORK');
    WRITELN;
    WRITELN('FROM DATA FILE THAT CONTAIN DATA OF ORIGINATOR AND DELAY NODE');
    WRITELN;
    HEADING;
    WRITELN;
    READLN(INFILE, ORIGSS); (* READ NUMBER OF ORIGINATOR NODE *)
    FOR INX := 1 TO ORIGSS DO
        BEGIN (* READ INFORMATION OF ORIGINATOR NODE *)
            READLN(INFILE, ORIC[INX].EXIT);
            READLN(INFILE, ORIC[INX].MINWAIT);
            READLN(INFILE, ORIC[INX].MAXWAIT);
        END;
    END;
```

```

READLN(INFILE,DECLASS); (* READ NUMBER OF DELAY NODE *)
READLN(INFILE,NOEXITSS); (* READ NUMBER OF EXITPATH FOR DELAY NODE *)
FOR INX := 1 TO DECLASS DO
  BEGIN (* READ INFORMATION OF DELAY NODE *)
    FOR INDX := 1 TO NOEXITSS DO
      BEGIN
        READLN(INFILE,DELC[INX].EXITSC[INDX]);
      END;
    FOR INDX := 1 TO NOEXITSS DO
      BEGIN
        READLN(INFILE,DELC[INX].EXPROB[INDX]);
      END;
    READLN(INFILE,DELC[INX].HOLDMIN);
    READLN(INFILE,DELC[INX].HOLDMAX);
  END;
END; (*OF READ DATA*)

C OUTPUTDATA:
  THIS PROCEDURE CONSTRUCT THE STATISTICAL TABLE FOR THIS COMMUNICATION
  NETWORK SYSTEM FOR 3 ORIGINATOR AND 7 DELAY NODES.
INPUT PARAMETERS :
  NONE
OUTPUT PARAMETERS :
  ORI      : 3 FIELDS OF 3 ORIGINATOR NODES.
  DEL      : 4 FIELDS OF 7 DELAY NODES. }

PROCEDURE OUTPUTDATA;
VAR
  K,M,N,IDX : INTEGER; (* USE FOR COUNTER NUMBER OF NODES *)
BEGIN
  UNDERLINE;
  WRITELN;
  HEADING;
  WRITELN; WRITELN;
  WRITELN('ORIGINATOR NODE NO. ':20,'EXIT':13,'MINWAIT':21,'MAXWAIT':24);
  WRITELN;
  HEADING;
  WRITELN;
  FOR N := 1 TO ORIG DO
    BEGIN (* WRITE INFORMATION OF ORIGINATOR NODES *)
      WRITELN(N:12,ORIC[N].EXIT:20,ORIC[N].MINWAIT:20:1,ORIC[N].MAXWAIT:24:1);
      UNDERLINE;
      WRITELN;
    END;
  HEADING;
  WRITELN;
  UNDERLINE;
  WRITELN; WRITELN; WRITELN; WRITELN; WRITELN; WRITELN;
  UNDERLINE;
  WRITELN;
  HEADING;
  WRITELN; WRITELN;
  WRITE('DELAY NODE NO. ':15,'EXITLIST':12,'EXPROBLIST':20,'HOLDMIN':17);
  WRITELN('HOLDMAX':15);
  WRITELN;
  HEADING;
  WRITELN;
  FOR M := 1 TO DELAS DO
    BEGIN (* WRITE INFORMATION OF DELAY NODES *)
      WRITE(M:7);
      WRITE(' ');
      FOR IDX := 1 TO NOEXITS DO
        WRITE(DELC[M].EXITSC[IDX]:3);
      WRITELN;
    END;
  END;

```

```

WRITE(' ');
FOR IDX := 1 TO NOEXITS DO
WRITE(DELIMJ,EXPROBCIDXJ:4:1);
WRITELN(DELIMJ,HOLDMIN:10:1,DELIMJ,HOLDMAX:15:1);
UNDERLINE;
WRITELN;
END;
HEADING;
WRITELN;
UNDERLINE;
WRITELN;
END; (*OUTPUTDATA*)

```

```

C INSERTEVENTQUEUE:
THIS PROCEDURE SEARCH THROUGH THE EVENTQUEUE TO DETERMINE WHERE A NEW
EVENT SHOULD BE PLACED. THEN ADD THIS NEW EVENT TO THE EVENTQUEUE,
KEEPING THE EVENT LIST ORDERED BY THE TIME FIELD.
INPUT PARAMETERS :
CALENDAR : POINTER TO A LIST OF 0 OR MORE EVENTS.
EVTIME : TIME THAT EVENT IS TO BE SCHEDULED IN EVENTQUEUE.
ACTIONTYPE: TYPE OF THE EVENT TO BE SCHEDULED.
NUMBER : NUMBER OF ORIGINATOR,DELAY OR DESTROYER NODE.
PPACK : POINTER TO THE PACKET INVOLVED [MAY BE NIL].
OUTPUT PARAMETERS :
CALENDAR : POINTER POINTS TO THE NEXT EVENT INTO EVENTQUEUE,
PRESERVING ORDERING BY THE TIME FIELD. }

```

```

PROCEDURE INSERTEVENTQUEUE(VAR CALENDAR:EVENTPTR;EVTIME:REAL;ACTIONTYPE:ACT;
NUMBER:INTEGER;PPACK:PTR);

```

```

VAR
ECCOUNT : INTEGER; (* COUNTER EVENTS IN EVENTQUEUE *)
EVEN : EVENTPTR; (* TEMPORARY FOR NODE CREATION *)
P,Q,R,PNODE : EVENTPTR; (* TEMPORARY USED FOR SEARCH IN EVENTQUEUE *)
BEGIN (*INSERT EVENTQUEUE*)
(* FIRST BUILD A NEW NODE WITH THE PROPER VALUE *)
NEW(EVEN);
WITH EVEN ^ DO
BEGIN
TIME := EVTIME;
ACTION := ACTIONTYPE;
PARM := NUMBER;
PACK := PPACK;
END; (* OF WITH *)
PNODE := CALENDAR;
(* NOW SEE WHERE THIS NEW NODE SHOULD BE PLACED AND RESET THE LINKS *)
IF PNODE <> NIL THEN
BEGIN
IF EVEQ.EFRONT = NIL THEN
BEGIN (* FIRST EVENT IN EVENTQUEUE *)
EVEQ.EFRONT := PNODE;
EVEQ.EBACK := PNODE;
PNODE^.LINK := NIL;
END
ELSE
IF EVEQ.EFRONT^.TIME > PNODE^.TIME THEN
BEGIN (*INSERT AT FRONT*)
PNODE^.LINK := EVEQ.EFRONT;
EVEQ.EFRONT := PNODE;
END
ELSE
BEGIN (*INSERT IN MIDDLE*)
P := EVEQ.EFRONT;
Q := EVEQ.EFRONT;

```



```

WHILE (P^.LINK <> NIL) AND (P=Q) DO
  BEGIN (*TRAVERSE*)
    P := P^.LINK;
    IF P^.TIME > PNODE^.TIME THEN
      BEGIN (*ATTACH*)
        Q^.LINK := PNODE;
        PNODE^.LINK := P;
      END (*ATTACH*)
    ELSE
      IF (P^.TIME = PNODE^.TIME) THEN
        BEGIN
          R := P^.LINK;
          P^.LINK := PNODE;
          PNODE^.LINK := R;
        END
      ELSE
        Q := P;
      END; (*TRAVERSE*)
    IF (P^.LINK = NIL) AND (P^.TIME <= PNODE^.TIME) THEN
      BEGIN (*ATTACH AT END*)
        P^.LINK := PNODE;
        EVEQ.EBACK := PNODE;
        PNODE^.LINK := NIL;
      END;
    END; (*INSERT IN MIDDLE*)
  END (* PNODE <> NIL *)
ELSE
  BEGIN (*PNODE = NIL*)
    PNODE := EVEN;
    IF EVEQ.EFRONT = NIL THEN
      BEGIN
        EVEQ.EFRONT := PNODE;
        EVEQ.EBACK := PNODE;
      END
    ELSE
      IF EVEQ.EFRONT^.TIME > PNODE^.TIME THEN
        BEGIN (*INSERT AT FRONT*)
          PNODE^.LINK := EVEQ.EFRONT;
          EVEQ.EFRONT := PNODE;
        END
      ELSE
        BEGIN (*INSERT IN MIDDLE*)
          P := EVEQ.EFRONT;
          Q := EVEQ.EFRONT;
          WHILE (P^.LINK <> NIL) AND (P=Q) DO
            BEGIN (*TRAVERSE*)
              P := P^.LINK;
              IF P^.TIME > PNODE^.TIME THEN
                BEGIN (*ATTACH*)
                  Q^.LINK := PNODE;
                  PNODE^.LINK := P;
                END (*ATTACH*)
              ELSE
                IF (P^.TIME = PNODE^.TIME) THEN
                  BEGIN
                    R := P^.LINK;
                    P^.LINK := PNODE;
                    PNODE^.LINK := R;
                  END
                ELSE
                  Q := P;
                END;
              END; (*TRAVERSE*)
            IF (P^.LINK = NIL) AND (P^.TIME <= PNODE^.TIME) THEN

```

```

        BEGIN (*ATTACH AT END*)
            P^.LINK := PNODE;
        END;
    END; (*INSERT IN MIDDLE*)
    END; (*PNODE = NIL*)
    EVEQ.ECOUN := EVEQ.ECOUN+1;(*UPDATE NUMBER OF EVENTS IN EVENTQUEUE*)
END; (*INSERTEVENTQUEUE*)

C GETNEXTEVENT:
    THIS PROCEDURE GET THE NEXT EVENT FROM EVENTQUEUE, REMOVING IT FROM THE
    FRONT OF EVENTQUEUE.
    INPUT PARAMETERS :
        CALENDAR : POINTER TO THE LIST OF 1 OR MORE EVENTS.
    OUTPUT PARAMETERS :
        CALENDAR : HAS HAD IT IS FIRST EVENT REMOVED.
        EETIME   : RETURNS THE TIME THIS EVENT IS TO TAKE PLACE.
        ACTIONTYPE: RETURNS THE TYPE OF THE EVENT.
        NUMBER   : RETURNS NUMBER OF NODE TYPE.
        PPACK    : RETURN A POINTER TO THE PACKET INVOLVED. }

PROCEDURE GETNEXTEVENT(VAR CALENDAR:EVENTPTR;VAR EETIME:REAL;VAR ACTIONTYPE:ACT;
VAR NUMBER:INTEGER;VAR PPACK:PTR);
VAR
    ECOUN : INTEGER; (* COUNTER EVENTS IN EVENTQUEUE *)
    DELETED : BOOLEAN;
BEGIN
    DELETED := FALSE;
    WITH EVEQ.EFRONT ^ DO
    BEGIN (* REMOVE FIRST EVENT IN EVENTQUEUE *)
        EETIME := TIME;
        ACTIONTYPE := ACTION;
        NUMBER := PARM;
        PPACK := PACK;
    END; (*OF WITH*)
    (* NOW ADJUST THE LINKS AND DISCARD THAT NODE *)
    EVEQ.EFRONT := EVEQ.EFRONT^.LINK;
    DELETED := TRUE;
    CALENDAR := EVEQ.EFRONT;
    EVEQ.ECOUN := EVEQ.ECOUN-1;(*UPDATE NUMBER OF EVENTS IN EVENTQUEUE*)
END; (*GET NEXT EVENT*)

C LISTEVENTQUEUE:
    THIS PROCEDURE LIST ALL OF THE EVENTS IN EVENTQUEUE.
    INPUT PARAMETERS :
        EV : POINTER TO THE FIRST EVENT IN EVENTQUEUE.
    OUTPUT PARAMETERS :
        TIME : SCHEDULE TIME OF EVENT IN EVENTQUEUE.
        NODID : ACTION TYPE OF EVENT.
        NODES : TYPE OF NODE IN THE SYSTEM.
        PARM : SPECIFY NUMBER OF NODE. }

PROCEDURE LISTEVENTQUEUE;
VAR
    EV : EVENTPTR; (*TEMPORARY USED FOR LIST ALL OF EVENTS IN EVENTQUEUE*)
BEGIN
    WRITELN;
    WRITELN('LIST OF THE EVENT IN EVENTQUEUE':56);
    WRITELN('-----':56);
    WRITELN;
    HEADING;
    WRITELN; WRITELN;
    WRITE('SCHEDULE TIME IN EVENTQUEUE':35,'ACTIONTYPE':18,'NODETYPE':17);
    WRITELN('NO.':8);

```

```

WRITE('-----':35,'-----':18,'-----':17);
WRITELN('---':8);
WRITELN;
HEADING;
WRITELN; WRITELN;
EV := EVEQ.EFRONT; (* SET POINTER TO THE FIRST EVENT IN EVENTQUEUE *)
WHILE EV <> NIL DO
BEGIN
  IF (EV.ACTION = CREATE) THEN
  BEGIN
    NODES := 'ORIGINATOR';
    NODED := 'CREATED';
  END;
  IF (EV.ACTION = LEAVE) THEN
  BEGIN
    NODES := 'DELAY';
    NODED := 'LEAVED';
  END;
  IF EV.ACTION = STOP THEN
  BEGIN
    NODES := 'STOP...';
    NODED := 'STOP...';
  END;
  WRITE(EV.TIME:22:3);
  WRITELN(NODED:32,NODES:17,EV.PARM:6);
  UNDERLINE;
  WRITELN;
  EV := EV.LINK; (* ADJUST POINTER TO THE NEXT EVENT *)
END;
WRITELN; WRITELN;
HEADING;
WRITELN; WRITELN;
WRITELN('EVENTS IN EVENTQUEUE =',EVEQ.ECOUNT:3);
WRITELN;
END; (*LISTEVENTQUEUE*)

```

(RANDOM:

THIS FUNCTION GET A RANDOM NUMBER [REAL NUMBERS] IN THE SPECIFIED RANGE FROM A UNIFORM DISTRIBUTION. THIS IS A SYSTEM DEPENDENT ROUTINE.

INPUT PARAMETERS :

LOW : LOWER BOUND OF RANGE.
HIGH : UPPER BOUND OF RANGE.

OUTPUT PARAMETERS :

THE FUNCTION RETURNS A NUMBER ON THE OPEN INTERVAL [LOW,HIGH]. }

FUNCTION RANDOM(LOW,HIGH:REAL):REAL;

FUNCTION RAN(VAR SEED : INTEGER) : REAL;

BEGIN

RAN := SEED/65535;
SEED := (25173*SEED+13849) MOD 65536;

END; (*RAN*)

BEGIN (*RANDOM*)

RANDOM := RAN(RANSEED)*(HIGH-LOW)+LOW;

END; (*FUNCTION RANDOM*)

(CREATES:

THIS PROCEDURE COMPUTES THE CREATION TIME OF NEXT PACKET, THIS TIME BASED ON DATA OF ORIGINATOR NODE [MINWAIT,MAXWAIT].

INPUT PARAMETERS :

ORINO : NUMBER OF ORIGINATOR NODE.

OUTPUT PARAMETERS :

```

        TIMES      : SET THE NEXT CREATION TIME IN CREATE PACKET SESSION. }

PROCEDURE CREATES(ORINO:INTEGER;VAR TIMES:REAL);
BEGIN
    TIMES := RANDOM(ORICORINOJ.MINWAIT,ORICORINOJ.MAXWAIT)+TT;
    ORICORINOJ.CURTIME := TIMES;
    (* ADJUST THE CURRENT TIME FOR ORIGINATOR NODE *)
END; (*CREATES*)

{ ARRIVES:
    THIS PROCEDURE COMPUTES WAITING TIME OF PACKET FROM DELAY NODE TO OTHER
    DELAY OR DESTROYER NODE, BASED ON DATA OF DELAY NODE [HOLDMIN,HOLDMAX].
    INPUT PARAMETERS :
        DELNOS      : NUMBER OF DELAY NODE.
    OUTPUT PARAMETERS :
        TIMES      : SET THE SERVICE TIME IN DELAY NODE. }

PROCEDURE ARRIVES(DELNOS:INTEGER;VAR TIMES:REAL);
BEGIN
    TIMES := RANDOM(DEL[DELNOS].HOLDMIN,DEL[DELNOS].HOLDMAX);
END; (*ARRIVES*)

{ EXITPATH:
    THIS PROCEDURE DETERMINE EXIT OF DELAY NODE IN ORDER TO SEND THE PACKET
    TO OTHER DELAY NODE OR DESTROYER NODE, BASED ON DATA OF DELAY NODE
    [EXITS,EXPROB].
    INPUT PARAMETERS :
        NOS        : NUMBER OF DELAY NODE.
    OUTPUT PARAMETERS :
        EXNO      : NUMBER OF EXITS OF THIS DELAY NODE. }

PROCEDURE EXITPATH(NOS:INTEGER;VAR EXNO:INTEGER);
VAR
    R : REAL; (* A PSEUDORANDOM NUMBER *)
    INDEX : INTEGER; (* COUNTER FOR NUMBER OF EXITPATH *)
    FLAGS : BOOLEAN; (* BOOLEAN TO COMPLETE THE EXECUTION *)
BEGIN
    FLAGS := TRUE;
    R := RANDOM(0.0,1.0);
    FOR INDEX := 1 TO NOEXITS DO
    BEGIN
        IF FLAGS = TRUE THEN
        BEGIN
            R := R - DEL[NOS].EXPROB[INDEX];
            IF R <= 0 THEN
            BEGIN
                EXNO := DEL[NOS].EXITS[INDEX];
                FLAGS := FALSE;
            END
            ELSE
            BEGIN
                FLAGS := TRUE;
            END;
        END;
    END;
END; (*EXITPATH*)

{ INSERT:
    THIS PROCEDURE INSERT A PACKET INTO QUEUE OF DELAY NODE WHEN THE PACKET
    ARRIVED DELAY NODE.
    INPUT PARAMETERS :
        DEL.Q      : POINTS TO A QUEUE OF 0 OR MORE PACKETS.
        NPTR      : POINTS TO PACKET BEING PLACED IN QUEUE.

```

```

OUTPUT PARAMETERS :
    DEL.Q.      : HAS HAD THE PACKET PUT AT THE END OF QUEUE.
    NPTR       : THE NEXT FIELD IN QUEUE IS SET TO NIL. }

PROCEDURE INSERT(VAR DEL:DELAY; NPTR:PTR);
BEGIN
    IF DEL.Q.FRONT = NIL THEN (* QUEUE IS EMPTY *)
    BEGIN
        DEL.Q.FRONT := NPTR;
        DEL.Q.BACK  := NPTR;
        NPTR^.NEXT  := NIL;
    END (* OF START QUEUE *)
    ELSE
    BEGIN (* QUEUE NOT EMPTY *)
        DEL.Q.BACK^.NEXT := NPTR;
        NPTR^.NEXT       := NIL;
        DEL.Q.BACK       := NPTR;
    END (* OF ADD TO NONEMPTY QUEUE *)
    DEL.Q.COUNT := DEL.Q.COUNT+1;
END; (*PROCEDURE INSERT IN QUEUE*)

{ DELETE:
    THIS PROCEDURE DELETE THE FIRST PACKET, NPTR, FROM QUEUE
INPUT PARAMETERS :
    DEL.Q      : POINTS TO A QUEUE OF 1 OR MORE PACKETS
OUTPUT PARAMETERS :
    NPTR       : POINTS TO THE FIRST PACKET IN QUEUE
    DEL.Q      : HAS HAD ITS FIRST PACKET REMOVED }

PROCEDURE DELETE(VAR NPTR:PTR;VAR DEL:DELAY);
BEGIN
    IF DEL.Q.FRONT <> NIL THEN (* THERE IS AT LEAST ONE PACKET IN THE QUEUE *)
    BEGIN
        NPTR := DEL.Q.FRONT;
        IF NPTR^.NEXT <> NIL THEN
            DEL.Q.FRONT := NPTR^.NEXT (* REMOVE FROM LIST *)
        ELSE
            BEGIN (* THE QUEUE HAD ONLY ONE PACKET AND IS NOW EMPTY *)
                DEL.Q.FRONT := NIL;
                DEL.Q.BACK  := NIL;
            END;
            DEL.Q.COUNT := DEL.Q.COUNT - 1;
        END (* OF QUEUE WAS NOT EMPTY *)
    END; (*PROCEDURE DELETE FROM QUEUE*)

{ ARRIVEDELAYNODE:
    THIS PROCEDURE SEND THE PACKET TO DELAY NODE, INSERT THAT PACKET IN
    DELAY QUEUE, COMPUTE LEAVING TIME FOR PACKET IN CASE DELAY QUEUE IS
    EMPTY OR NOT EMPTY, ALSO CHANGED THE STATE OF EVENT FROM CREATE TO
    LEAVE EVENT OR FROM LEAVE EVENT TO ANOTHER LEAVE EVENT, THEN INSERT
    INTO THE EVENTQUEUE
INPUT PARAMETERS :
    DELNO      : NUMBER OF DELAY NODE
    NPTR       : POINTS TO THE PACKET THAT JUST EMANATED TO DELAY NODE
    DEL        : POINTER TO SPECIFY DELAY NODE NUMBER
OUTPUT PARAMETERS :
    TEMP1      : LEAVING TIME FROM DELAY NODE, IN CASE QUEUE EMPTY
    TEMP2      : LEAVING TIME FROM DELAY NODE, IN CASE QUEUE NOT EMPTY }

PROCEDURE ARRIVEDELAYNODE(VAR DELNO:INTEGER;VAR NPTR:PTR;VAR DEL:DELA;
    VAR TEMP1:REAL;VAR TEMP2:REAL);
VAR
    T : REAL; (* PSEUDORANDOM NUMBER *)

```

```

BEGIN
  DELCDELNOJ.COUNT := DELCDELNOJ.COUNT+1; (*COUNT PACKET IN DELAY NODE*)
  IF DELCDELNOJ.Q.FRONT = NIL THEN
    BEGIN (*IF DELAY QUEUE EMPTY*)
      INSERT(DELNO,T);
      ARRIVES(DELNO,T); (* COMPUTE SERVICE TIME *)
      TEMP1 := T+TT; (* LEAVING TIME FOR PACKET *)
      DELCDELNOJ.LAST := TEMP1;
      NPTR^.WAIT := NPTR^.WAIT+TEMP1-TT-T;
      NPTR^.MARKS := NPTR^.MARKS+TEMP1-TT;
      DELCDELNOJ.WAITED := DELCDELNOJ.WAITED+TEMP1-TT;
      DELCDELNOJ.WAIT := DELCDELNOJ.WAIT+TEMP1-TT-T;
      DELCDELNOJ.MARK := DELCDELNOJ.MARK+TEMP1-TT;
      (* SCHEDULE THE LEAVE EVENT AT TIME TEMP1 INTO EVENTQUEUE *)
      INSERTEVENTQUEUE(DELNO,CALENDAR,TEMP1,LEAVE,DELNO,NPTR);
    END
  ELSE
    BEGIN (*IF DELAY QUEUE NOT EMPTY*)
      ARRIVES(DELNO,T); (* COMPUTE SERVICE TIME *)
      (* SET LEAVING TIME FOR THAT PACKET FROM DELAY NODE *)
      TEMP2 := DELCDELNOJ.LAST+T;
      DELCDELNOJ.LAST := TEMP2;
      NPTR^.MARKS := NPTR^.MARKS+TEMP2-TT;
      NPTR^.WAIT := NPTR^.WAIT+TEMP2-TT-T;
      DELCDELNOJ.WAITED := DELCDELNOJ.WAITED+TEMP2-TT;
      DELCDELNOJ.WAIT := DELCDELNOJ.WAIT+TEMP2-TT-T;
      NPTR^.NEX := TEMP2;
      INSERT(DELNO,T);
      (* UPDATE TOTAL TIME THAT WAIT IN DELAY NODE *)
      DELCDELNOJ.MARK := DELCDELNOJ.MARK+TEMP2-TT;
      (* SCHEDULE LEAVE EVENT AT TIME TEMP2 INTO EVENTQUEUE *)
      INSERTEVENTQUEUE(DELNO,CALENDAR,TEMP2,LEAVE,DELNO,NPTR);
    END;
  END; (*OF ARRIVE DELAY NODE*)

(* ARRIVEDESTROYNODE:
  THIS PROCEDURE SEND THE PACKET TO DESTROYER NODE, COMPUTE THE TRANSIT
  TIME FOR THAT PACKET AND UPDATE THE TOTAL TRANSIT TIME IN DESTROYER
  NODE
  INPUT PARAMETERS :
  DESNO : NUMBER OF DESTROYER NODE
  NPTR : POINTS TO THE PACKET THAT JUST EMANATED TO DESTROYER NODE
  DES : POINTER TO SPECIFY DESTROYER NODE
  OUTPUT PARAMETERS :
  NPTR : DELETE THIS PACKET FROM THE SYSTEM *)

PROCEDURE ARRIVEDESTROYNODE(VAR DESNO:INTEGER;VAR NPTR:PTR;VAR DES:DEST);
BEGIN
  DESCDESNOJ.COUNT := DESCDESNOJ.COUNT+1;
  DESCDESNOJ.TRANSIT := TT-NPTR^.BORN;
  DESCDESNOJ.TOTALTIME := DESCDESNOJ.TOTALTIME+DESCDESNOJ.TRANSIT;
END; (* OF ARRIVE DESTROY NODE *)

(* CREATEPACKET:
  THIS PROCEDURE HANDLE A PACKET THAT JUST CREATED,SCHEDULE NEXT CREATE
  TIME IN EVENTQUEUE AND THEN SEND THAT PACKET TO DELAY NODE OR DESTROYER
  NODE. THIS PROCEDURE AUTOMATICALLY CREATE NEW PACKET AT ORIGINATOR NODE
  AFTER THE FORMER PACKET IS EMANATED TO DELAY NODE
  INPUT PARAMETERS :
  NN : SPECIFIED NUMBER OF ORIGINATOR NODE
  ORI : POINTER TO THE SPECIFY ORIGINATOR NODE
  DEL : POINTER TO THE SPECIFY DELAY NODE
  OUTPUT PARAMETERS :

```

```

DELNO      : SPECIFIED NUMBER OF EXIT FROM ORIGINATOR NODE
TIMSO      : NEXT CREATE PACKET TIME FOR ORIGINATOR NODE
TIMS1      : LEAVING TIME OF THE PACKET FROM DELAY NODE IN CASE DELAY
             QUEUE IS EMPTY
TIMS2      : LEAVING TIME OF THE PACKET FROM DELAY NODE IN CASE DELAY
             QUEUE IS NOT EMPTY }

```

```

PROCEDURE CREATEPACKET(VAR ORI:ORIG;VAR NN:INTEGER;VAR TIMSO:REAL;
                       VAR DELNO:INTEGER;VAR TIMS1:REAL;VAR DEL:DELA;
                       VAR DES:DEST;VAR TIMS2:REAL);

```

```

VAR

```

```

T          : REAL; (* A PSEUDORANDOM NUMBER *)
T1,T2     : REAL; (* LEAVING TIME FROM DELAY NODE *)
OO        : INTEGER; (* TEMPORARY NUMBER OF ORIGINATOR NODE *)

```

```

BEGIN

```

```

ORICNNJ.COUNT := ORICNNJ.COUNT+1; (* COUNT NUMBER OF PACKET CREATED *)
CREATES(NN,T); (* COMPUTE NEXT CREATION TIME *)
OO := NN;
TIMSO := T;
(* SCHEDULE NEXT CREATION TIME IN TIMSO MIN. INTO EVENTQUEUE *)
INSERTEVENTQUEUE(ORICOOJ.CALENDAR,TIMSO,CREATE,OO,NIL);
(* NOW CREATE THAT NEW PACKET AND SET INITIAL PARAMETERS *)
NEW(NPTR);
WITH NPTR DO
BEGIN
  BORN := TT;
  ORG := OO;
  MARKS := 0.0;
  WAIT := 0.0;
  NEX := 0.0;
  PACNO := ORICOOJ.COUNT;
END;
DELNO := ORICOOJ.EXIT; (* NUMBER EXIT FROM ORIGINATOR NODE *)
(* CONSIDER SIGN OF NUMBER EXIT, IF POSITIVE NO. MEAN DELAY NODE ELSE
   MEAN DESTROYER NODE *)
IF DELNO > 0 THEN
BEGIN (* PACKET IS SENT TO DELAY NODE *)
  ARRIVEDELAYNODE(DELNO,NPTR,DELS,T1,T2);
  TIMS1 := T1;
  TIMS2 := T2;
END
ELSE
BEGIN (* PACKET IS SENT TO DESTROYER NODE *)
  ARRIVEDESTROYNODE(DELNO,NPTR,DESS);
END;
END; (*CREATEPACKET*)

```

```

( LEAVEDELAY:

```

```

THIS PROCEDURE TRANSFERS PACKET FROM ONE DELAY NODE TO ANOTHER DELAY OR
DESTROYER NODE BY COMPUTE THE EXITPATH.

```

```

INPUT PARAMETERS :

```

```

DELNUM      : NUMBER OF DELAY NODE
NPTR        : POINTS TO THE PACKET THAT IS REMOVED
DEL         : POINTER TO SPECIFIC DELAY NODE
DES         : POINTER TO SPECIFIC DESTROYER NODE

```

```

OUTPUT PARAMETERS :

```

```

EXNOS      : SPECIFIED NUMBER OF EXITS OF DELAY NODE
TEMB       : TIME LEAVING FROM DELAY NODE, IN CASE QUEUE EMPTY
TEMB1      : TIME LEAVING FROM DELAY NODE, IN CASE QUEUE NOT EMPTY }

```

```

PROCEDURE LEAVEDELAY(VAR NPTR:PTR;VAR DELNUM:INTEGER;VAR DEL:DELA;VAR DES:DEST;
                    VAR EXNOS:INTEGER;VAR TEMB:REAL;VAR TEMB1:REAL);

```

```

VAR

```

```

TEM,TEMS : REAL; (* LEAVING TIME FROM DELAY NODE *)
MM      : INTEGER; (* TEMPORARY NUMBER OF EXIT *)
BEGIN
EXITPATH(DELNUM,MM); (* COMPUTE EXIT FROM DELAY NODE *)
EXNOS := MM; (* NUMBER OF EXIT AFTER COMPUTE EXITPATH *)
(* NOW REMOVE PACKET FROM QUEUE OF DELAY NODE *)
DELETE(NPTR,DEL[DELNUM]);
(* CONSIDER SIGN OF EXIT, POSITIVE NO.MEAN DELAY NODE ELSE MEAN
DESTRUCTOR NODE *)
IF EXNOS > 0 THEN
BEGIN (*SENT PACKET TO DELAY NODE*)
ARRIVEDELAYNODE(EXNOS,NPTR,DELS,TEM,TEMS);
TEMB := TEM; (* LEAVING TIME FROM DELAY NODE, QUEUE EMPTY *)
TEMB1:= TEMS; (* LEAVING TIME FROM DELAY NODE, QUEUE NOT EMPTY *)
END
ELSE
BEGIN (*SENT PACKET TO DESTROYER NODE*)
ARRIVEDESTROYNODE(EXNOS,NPTR,DESS);
END;
END; (*LEAVEDELAY*)

{ STOPSIM:
THIS PROCEDURE HANDLE EVENT THAT CURRENT TIME > MAXIMUM TIME FOR RUN
THIS PROGRAM BY HALTING THE PROGRAM
INPUT PARAMETERS :
NONE
OUTPUT PARAMETERS :
NONE
THE PROGRAM WILL BE HALTED }

PROCEDURE STOPSIM;
BEGIN
WRITELN; WRITELN; WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN;
WRITE('*****');
HEADING;
WRITELN; WRITELN; WRITELN;
WRITELN('CURTIME IS GREATER THAN MAXTIME':55);
WRITELN;
WRITELN('WE WILL STOP THE SYSTEM SIMULATION AT THIS TIME':63);
WRITELN;
WRITELN('THESE ARE THE EVENTS THAT PENDING IN EVENTQUEUE':63);
WRITELN; WRITELN;
HEADING;
WRITELN;
UNDERLINE;
WRITELN; WRITELN; WRITELN;
END; (*STOPSIM*)

{ INITIALORI:
THIS PROCEDURE INITIALIZE THE STATE OF ORIGINATOR NODE AND ZERO OUT
STATISTICS
INPUT PARAMETERS :
NONE
OUTPUT PARAMETERS :
ORI      : ALL FIELDS ARE INITIALIZED OR SET TO ZERO EXCEPT EXIT,
MINWAIT,MAXWAIT WHICH HAVE BEEN READ FROM INPUT DATA FILE }

PROCEDURE INITIALORI(VAR ORI:ORIGINATOR);
BEGIN
WITH ORI DO

```



```

        BEGIN
            CURTIME := 0.0;
            COUNT := 0;
            INSERTEVENTQUEUE(CALENDAR,STARTTIME,CREATE,NO,NIL);
        END; (*OF WITH ORI*)
    END; (*INITIALORI*)

C INITIALDEL:
    THIS PROCEDURE INITIALIZE THE STATE OF DELAY NODE AND ZERO OUT
    STATISTICS
    INPUT PARAMETERS :
        NONE
    OUTPUT PARAMETERS :
        DEL      : ALL FIELDS ARE INITIALIZED OR SET TO ZERO EXCEPT EXITS,
                  EXPROB,HOLDMIN,HOLDMAX,AND MAX WHICH HAVE BEEN READ FROM
                  INPUT DATA FILE }

PROCEDURE INITIALDEL(VAR DEL:DELAY);
BEGIN
    WITH DEL DO
    BEGIN
        Q.FRONT := NIL;
        Q.BACK  := NIL;
        Q.COUNT := 0;
        COUNT   := 0;
        MARK    := 0.0;
        AVER    := 0.0;
        WAITED  := 0.0;
        WAIT    := 0.0;
        LAST    := 0.0;
    END; (*OF WITH DEL*)
END; (*INITIALDEL*)

C INITIALDES:
    THIS PROCEDURE INITIALIZE THE STATE OF DESTROYER NODE AND ZERO OUT
    STATISTICS
    INPUT PARAMETERS :
        NONE
    OUTPUT PARAMETERS :
        DES      : ALL FIELDS ARE SET TO ZERO }

PROCEDURE INITIALDES(VAR DES:DESTROYER);
BEGIN
    WITH DES DO
    BEGIN
        TRANSIT := 0.0;
        TOTALTIME := 0.0;
        COUNT := 0;
    END; (*OF WITH DES*)
END; (*INITIALDES*)

C REPORTHEADING:
    THIS PROCEDURE CONSTRUCT FORMAT HEADING FOR REPORTED ACTION OF EVENT
    WHEN GET AND PERFORMED ACTION OF EVENT FROM EVENTQUEUE
    INPUT PARAMETERS :
        NONE
    OUTPUT PARAMETERS :
        NONE
        THE FORMAT HEADING WILL BE WRITTEN IF THE CONDITION TEST IS ACCEPTED }

PROCEDURE REPORTHEADING;
BEGIN
    WRITELN; WRITELN; WRITELN; WRITELN; WRITELN; WRITELN; WRITELN;

```

```

WRITELN; WRITELN; WRITELN; WRITELN; WRITELN; WRITELN;
WRITE('*****');
HEADING;
WRITELN; WRITELN;
WRITELN('THIS IS A FORMAT HEADING FOR REPORT ACTION OF EVENTS':92);
WRITELN('-----':92);
WRITELN;
WRITELN('IN CASE OF ACTION = CREATE : [ O => DS ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT CREATE':18);
WRITELN('TIME LEAVING':40);
WRITELN;
WRITELN('IN SYSTEM':30,'DESCRIPTOR':13);
WRITE('*****');
HEADING;
WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN; WRITELN;
WRITELN('IN CASE OF ACTION = CREATE : [ O => DQ ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT CREATE':18);
WRITELN('TIME LEAVING':40,'Q-SIZE':10);
WRITELN;
WRITELN('IN SYSTEM':30,'DESCRIPTOR':13,'TONODE':65);
WRITE('*****');
HEADING;
WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN; WRITELN;
WRITELN('IN CASE OF ACTION = ORIGINATOR ==> DESTROYER : [ O => K ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT CREATE':18);
WRITELN('TOTAL':62,'TRANSIT':11);
WRITELN;
WRITELN('IN SYSTEM':30,'DESCRIPTOR':13,'KILLED':78,'TIME':9);
WRITE('*****');
HEADING;
WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN; WRITELN;
WRITELN('IN CASE OF ACTION = DELAY ==> DELAY : [ D => DS ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT EV':31);
WRITELN('Q-SIZE':12,'TIME LEAVING':15);
WRITELN;
WRITELN('IN SYSTEM':30,'DESCRIPTOR':13,'FROMNODE':29,'FROMNODE':12);
WRITE('*****');
HEADING;

```

```

WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN; WRITELN;
WRITELN('IN CASE OF ACTION = DELAY ==> DELAY : [ D => DQ ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT EV':31);
WRITELN('Q-SIZE':12,'TIME LEAVING':15,'Q-SIZE':10);
WRITELN;
WRITE('IN SYSTEM':30,'DESCRIPTOR':13,'FROMNODE':29,'FROMNODE':12);
WRITELN('TONODE':24);
WRITE('*****');
HEADING;
WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN; WRITELN;
WRITELN('IN CASE OF ACTION = DELAY ==> DESTROYER : [ D => K ]');
WRITELN;
WRITE('*****');
HEADING;
WRITELN;
WRITE('TIME':6,'PAC.#':11,'PAC. TIME':13,'EVENT':10,'NEXT EV':31);
WRITELN('Q-SIZE':12,'TOTAL':37,'TRANSIT':11);
WRITELN;
WRITE('IN SYSTEM':30,'DESCRIPTOR':13,'FROMNODE':29,'FROMNODE':12);
WRITELN('KILLED':37,'TIME':9);
WRITE('*****');
HEADING;
WRITELN;
WRITE('-----');
UNDERLINE;
WRITELN;
END; (*REPORT HEADING*)

```

REPORT:

THIS PROCEDURE REPORTED AFTER PERFORMED THE ACTION OF THE EVENT. IT WILL BE REPORTED EVERY TIMES WHEN THE EVENT IS GET FROM EVENTQUEUE IF THE CONDITION TEST IS ACCEPTED. THIS REPORT WILL PRINT INFORMATION AS FOLLOW:-

- CURRENT TIME[CLOCK] OF EVENT
- NUMBER OF PACKET AND ORIGINATOR NODE THAT CREATED
- TIME THAT PACKET BORN
- EVENT DESCRIPTOR
- NEXT CREATION TIME
- NEXT LEAVING TIME FROM NODE
- Q-SIZE FROM DELAY NODE
- LEAVING TIME FROM NODE
- Q-SIZE TO NODE
- TOTAL PACKET DESTROYED
- TRANSIT TIME FOR EACH PACKET

INPUT PARAMETERS :

```

A      : ACTIONTYPE OF EVENT
TT     : TIME OF OCCURENCE EVENT
NO3    : NUMBER OF EXIT FROM ORIGINATOR NODE
NO2    : NUMBER OF EXIT FROM DELAY NODE
PARM   : SPECIFY NUMBER OF NODE IN THE SYSTEM
DELS   : SPECIFY DELAY NODE NUMBER FROM ARRAY OF DELAY NODE RECORD
DESS   : SPECIFY DESTROYER NODE NUMBER FROM ARRAY OF DESTROYER NODE

```

```

                                RECORD
OUTPUT PARAMETERS :
  NONE
  THE REPORT WILL BE WRITTEN AFTER GET AND PERFORMED ACTION FROM
  EVENTQUEUE }

PROCEDURE REPORT(A:ACT);
VAR
  SIZE,SIZE1 : INTEGER; (* Q-SIZE OF DELAY QUEUE *)
  ANS,ANSW : CHAR;
BEGIN
  IF A = CREATE THEN
    BEGIN (* ACTION TYPE = CREATE *)
      IF NO3 > 0 THEN
        BEGIN
          ANS := '-';
          SIZE1 := DELSCNO3].Q.COUNT;
          IF DELSCNO3].Q.COUNT = 1 THEN
            BEGIN
              WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
              WRITELN('D':6,NPTR^.ORG:2,'=>DS',NO3:2,TIM:13:3,TEM4:39:3);
            END
          ELSE
            BEGIN
              WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
              WRITE('D':6,NPTR^.ORG:2,'=>DQ',NO3:2,TIM:13:3,TEM6:39:3);
              WRITELN(SIZE1:14);
            END;
          END;
        END;
      ELSE
        BEGIN
          WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
          WRITE('D':6,NPTR^.ORG:2,'=>K',ABS(NO3):2,TIM:13:3);
          WRITELN(DESSCNO3].COUNT:62,DESSCNO3].TRANSIT:13:3);
        END;
      END;
    IF A = LEAVE THEN
      BEGIN (* ACTION TYPE = LEAVE *)
        IF NO2 > 0 THEN
          BEGIN
            IF DELSCNO2].Q.COUNT = 1 THEN
              BEGIN
                ANS := 'E';
                ANSW := '-';
                SIZE := DELSCPARM].Q.COUNT;
                SIZE1 := 0;
                IF DELSCPARM].Q.COUNT = 0 THEN
                  BEGIN
                    WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
                    WRITELN('D':6,PARM:2,'=>DS',NO2:2,ANS:25,SIZE:13,TEM2:14:3);
                  END
                ELSE
                  BEGIN
                    WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
                    WRITE('D':6,PARM:2,'=>DS',NO2:2);
                    WRITELN(DELSCPARM].Q.FRONT^.NEX:28:3,SIZE:10,TEM2:14:3);
                  END;
                END;
              END
            ELSE
              BEGIN
                ANS := 'E';
                ANSW := '-';
                SIZE := DELSCPARM].Q.COUNT;
              END
          END
        END
      END
    END
  END

```

```

        SIZE1 := DELSCNO2].Q.COUNT;
        IF DELSCPARM].Q.COUNT = 0 THEN
        BEGIN
            WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
            WRITE('D':6,PARM:2,'=>DQ',NO2:2);
            WRITELN(ANS:25,SIZE:13,TEM3:14:3,SIZE1:14);
        END
        ELSE
        BEGIN
            WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
            WRITE('D':6,PARM:2,'=>DQ',NO2:2);
            WRITE(DELSCPARM].Q.FRONT^.NEX:28:3,SIZE:10);
            WRITELN(TEM3:14:3,SIZE1:14);
        END;
    END;
END;
ELSE
BEGIN
    SIZE := DELSCPARM].Q.COUNT;
    ANS := 'E';
    IF SIZE = 0 THEN
    BEGIN
        WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
        WRITE('D':6,PARM:2,'=> K',ABS(NO2):2,ANS:25);
        WRITELN(SIZE:13,DESSCNO2].COUNT:37,DESSCNO2].TRANSIT:13:3);
    END
    ELSE
    BEGIN
        WRITE(TT:7:3,NPTR^.ORG:7,',',NPTR^.PACNO:2,NPTR^.BORN:12:3);
        WRITE('D':6,PARM:2,'=> K',ABS(NO2):2);
        WRITE(DELSCPARM].Q.FRONT^.NEX:28:3);
        WRITELN(SIZE:10,DESSCNO2].COUNT:37,DESSCNO2].TRANSIT:13:3);
    END;
END;
END;
IF A = STOP THEN
BEGIN
    WRITELN;
    UNDERLINE;
    WRITELN;
    HEADING;
    WRITELN;
    WRITE('*')
    WRITELN('')
    WRITE('*')
    WRITELN('')
    WRITE('*')
    WRITELN('')
    WRITE('*')
    WRITELN('PROGRAM')
    WRITE('*')
    WRITELN('')
    WRITE('*')
    WRITELN('')
    WRITE('*')
    WRITELN('')
    HEADING;
    WRITELN;
    UNDERLINE;
    WRITELN; WRITELN;
END;
END; (*REPORT*)

C FINALREPORT:
THIS PROCEDURE PRODUCED THE FINISHED REPORT OF THE SIMULATION NETWORK
SYSTEM. THIS REPORT WILL PRINT ALL INFORMATION AS FOLLOW :-
- TOTAL PACKETS CREATED BY EACH ORIGINATOR NODE

```

- TOTAL PACKETS CREATED BY ALL ORIGINATOR NODES
- TOTAL PACKET FLOW THROUGH EACH DELAY NODE
- TOTAL PACKETS IN DELAY NODES
- AVERAGE DELAY TIME AT EACH NODE
- TOTAL DELAY TIME IN DELAY NODES
- AVERAGE DELAY TIME IN DELAY NODE
- TOTAL PACKET DESTROYED BY EACH DESTROYER NODE
- AVERAGE TRANSIT TIME IN EACH DESTROYER NODE
- TOTAL PACKET DESTROYER BY ALL DESTROYER NODES
- TOTAL LIFETIME OF DESTROYER NODES
- AVERAGE TRANSIT TIME

INPUT PARAMETERS:

ORI : ARRAY OF ALL ORIGINATOR NODE RECORDS
 DEL : ARRAY OF ALL DELAY NODE RECORDS
 DES : ARRAY OF ALL DESTROYER NODE RECORDS

OUTPUT PARAMETERS:

NONE
 THE RESULTS OF THIS PROGRAM WILL BE WRITTEN TO OUTPUT FILE }

PROCEDURE FINALREPORT(ORI:ORIG;DEL:DELA;DES:DEST);

VAR

K,L,C,B,X,Y : INTEGER; (* INDEX FOR NODE TYPE *)
 TOL : INTEGER; (* TOTAL PACKETS CREATED BY ORIGINATOR NODES *)
 TOT : INTEGER; (* TOTAL PACKET IN DELAY NODES *)
 TOLS : INTEGER; (* TOTAL PACKET DESTROYED BY DESTROYER NODES *)
 TOLT : REAL;(* TOTAL TRANSIT TIME FOR ALL PACKETS IN DESTROYER NODES *)
 TOTA : REAL;(* TOTAL DELAY TIME FOR ALL PACKETS IN DELAY NODES *)
 AVG : REAL; (* AVERAGE DELAY TIME IN DELAY NODE *)
 AVE : REAL; (* AVERAGE TRANSIT TIME FOR ALL PACKETS IN EACH
 DESTROYER NODE *)
 AVES : REAL; (* AVERAGE TRANSIT TIME FOR PACKET *)
 WAITING : REAL;

BEGIN

WRITELN#
 WRITELN(OUTFILE); WRITELN(OUTFILE);
 WRITELN(OUTFILE,'RESULTS OF THE SIMULATION RUN FOR NETWORK SYSTEM':62);
 WRITELN(OUTFILE,'=====':62);
 WRITELN(OUTFILE,'*****':62);
 WRITELN(OUTFILE); WRITELN(OUTFILE);
 WRITE(OUTFILE,'*****');
 WRITELN(OUTFILE,'*****');
 WRITELN(OUTFILE); WRITELN(OUTFILE);
 WRITELN(OUTFILE,'ORIGINATOR NODE':30,'TOTAL PACKET CREATED':35);
 WRITELN(OUTFILE,'-----':30,'-----':35);
 WRITELN(OUTFILE); WRITELN(OUTFILE);
 WRITE(OUTFILE,'*****');
 WRITELN(OUTFILE,'*****');
 WRITELN(OUTFILE); WRITELN(OUTFILE);
 FOR L := 1 TO ORIG DO
 BEGIN
 WRITELN(OUTFILE,L:23,ORICLJ.COUNT:33);
 WRITE(OUTFILE,'-----');
 WRITELN(OUTFILE,'-----');
 END;
 TOL := 0; (* INITIAL VALUE FOR TOL *)
 FOR C := 1 TO ORIG DO
 BEGIN
 TOL := TOL+ORICCJ.COUNT;
 END;
 WRITELN(OUTFILE); WRITELN(OUTFILE); WRITELN(OUTFILE);
 WRITELN(OUTFILE,'TOTAL PACKETS CREATED BY ORIGINATOR NODES =',TOL:10);
 WRITELN(OUTFILE); WRITELN(OUTFILE); WRITELN(OUTFILE);

```

WRITE(OUTFILE,'*****');
Writeln(OUTFILE,'*****');
WRITE(OUTFILE,'*****');
Writeln(OUTFILE,'*****');
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE,'*****');
Writeln(OUTFILE,'*****');
Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE,'DELAY NODE':12,'TOTAL PACKET IN DELAY NODE':29);
Writeln(OUTFILE,'AVERAGE DELAY TIME AT NODE':29);
WRITE(OUTFILE,'-----':12,'-----':29);
Writeln(OUTFILE,'-----':29);
Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE,'*****');
Writeln(OUTFILE,'*****');
Writeln(OUTFILE); Writeln(OUTFILE);
FOR X := 1 TO DELAS DO
BEGIN
  IF (DEL[X].COUNT <> 0) AND (DEL[X].MARK <> 0) THEN
  BEGIN
    DEL[X].AVER := (DEL[X].MARK)/(DEL[X].COUNT);
    WAITING     := (DEL[X].WAIT)/(DEL[X].COUNT);
  END
  ELSE
  BEGIN
    DEL[X].AVER := 0.0;
  END;
  Writeln(OUTFILE,X:9,DEL[X].COUNT:18,WAITING:31:3);
  WRITE(OUTFILE,'-----');
  Writeln(OUTFILE,'-----');
END;
TOT := 0; (* INITIAL VALUE FOR TOT *)
TOTA := 0.0; (* INITIAL VALUE FOR TOTA *)
FOR Y := 1 TO DELAS DO
BEGIN
  TOT := TOT+DEL[Y].COUNT;
  TOTA := TOTA+DEL[Y].WAIT;
END;
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
Writeln(OUTFILE,'TOTAL PACKETS IN DELAY NODES      =',TOT:10);
Writeln(OUTFILE); Writeln(OUTFILE);
Writeln(OUTFILE,'TOTAL DELAY TIME IN DELAY NODES    =',TOTA:10:3);
Writeln(OUTFILE);
IF TOT = 0 THEN
BEGIN
  Writeln(OUTFILE,'-----':62);
  Writeln(OUTFILE,'*****':62);
  Writeln(OUTFILE);
  Writeln(OUTFILE,'CAN NOT FIND AVERAGE DELAY TIME IN DELAY NODE':62);
  Writeln(OUTFILE);
  Writeln(OUTFILE,'*****':62);
  Writeln(OUTFILE,'-----':62);
  AVG := 0.0;
END
ELSE
BEGIN
  AVG := TOTA/TOT;
END;
Writeln(OUTFILE);
Writeln(OUTFILE,'AVERAGE DELAY TIME IN DELAY NODE =',AVG:10:3);
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE,'*****');
Writeln(OUTFILE,'*****');

```

```

WRITE(OUTFILE, '*****');
Writeln(OUTFILE, '*****');
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE, '*****');
Writeln(OUTFILE, '*****');
Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE, 'DESTROY NODE':20, 'TOTAL PACKET DESTROYED':27);
Writeln(OUTFILE, 'AVERAGE TIME':17);
WRITE(OUTFILE, '-----':20, '-----':27);
Writeln(OUTFILE, '-----':17);
Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE, '*****');
Writeln(OUTFILE, '*****');
Writeln(OUTFILE); Writeln(OUTFILE);
FOR K := -1 DOWNT0 DESTS DO
BEGIN
  IF DESCKJ.COUNT <> 0 THEN
  BEGIN
    AVE := DESCKJ.TOTALTIME/DESCKJ.COUNT;
  END
  ELSE
  BEGIN
    AVE := 0.0;
  END;
  Writeln(OUTFILE, ABS(K):15, DESCKJ.COUNT:22, AVE:23:3);
  WRITE(OUTFILE, '-----');
  Writeln(OUTFILE, '-----');
END;
TOLS := 0; (* INITIAL VALUE FOR TOLS *)
TOLT := 0.0; (* INITIAL VALUE FOR TOLT *)
FOR B := -1 DOWNT0 DESTS DO
BEGIN
  TOLS := TOLS+DESCBJ.COUNT;
  TOLT := TOLT+DESCBJ.TOTALTIME;
END;
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
Writeln(OUTFILE, 'TOTAL PACKET DESTROYED BY DESTROYER NODES =', TOLS:10);
Writeln(OUTFILE); Writeln(OUTFILE);
Writeln(OUTFILE, 'TOTAL LIFETIME OF PACKETS =', TOLT:10:3);
Writeln(OUTFILE);
IF TOLS = 0 THEN
BEGIN
  Writeln(OUTFILE, '-----':62);
  Writeln(OUTFILE, '*****':62);
  Writeln(OUTFILE);
  Writeln(OUTFILE, 'CAN NOT FIND THE AVERAGE TRANSIT TIME':62);
  Writeln(OUTFILE);
  Writeln(OUTFILE, '*****':62);
  Writeln(OUTFILE, '-----':62);
  AVES := 0.0;
END
ELSE
BEGIN
  AVES := TOLT/TOLS;
END;
Writeln(OUTFILE);
Writeln(OUTFILE, 'AVERAGE TRANSIT TIME =', AVES:10:3);
Writeln(OUTFILE); Writeln(OUTFILE); Writeln(OUTFILE);
WRITE(OUTFILE, '*****');
Writeln(OUTFILE, '*****');
WRITE(OUTFILE, '*****');
Writeln(OUTFILE, '*****');
END; (*OF FINAL REPORT*)

```



```

C INITSYS:
  THIS PROCEDURE INITIAL THE SYSTEM SIMULATION, CONSTRUCT DATA TABLE FOR
  ORIGINATOR AND DELAY NODE. READ THE INITIAL VALUE OF SEED NUMBER TO
  GENERATE RANDOM NUMBER. THIS PROCEDURE ALSO INITIAL VALUE FOR ALL
  ORIGINATOR, DELAY AND DESTROYER NODES
  INPUT PARAMETERS :
    RANSEED : NUMBER OF SEED VALUE USE FOR GENERATE RANDOM NUMBER
  OUTPUT PARAMETERS :
    NONE }

PROCEDURE INITSYS;
VAR
  WW : INTEGER; (*REPRESENT NUMBER OF NODE*)
BEGIN
  WRITELN; WRITELN; WRITELN;
  WRITELN('TABLES DESCRIBING THE NETWORK SYSTEM':58);
  WRITELN('-----':58);
  WRITELN; WRITELN; WRITELN;
  OUTPUTDATA;
  WRITELN;
  WRITELN('PLEASE ENTER THE VALUE OF SEED FOR INITIAL FIRST RANDOM NO. ');
  WRITE('SHOULD BE INTEGER NUMBER !!!! : ');
  READLN(RANSEED);
  WRITELN(RANSEED);
  WRITELN; WRITELN;
  WW := 0; (* SET VALUE OF WW *)
  EVEQ.EFRONT := NIL;
  EVEQ.EBACK := NIL;
  EVEQ.ECOUNT := 0;
  FOR NO := 1 TO ORIG$ DO (* INITIAL ALL ORIGINATOR NODES *)
  BEGIN
    INITIALORI(ORICNO);
  END;
  FOR XX := 1 TO DELAS DO (* INITIAL ALL DELAY NODES *)
  BEGIN
    INITIALDEL(DELSCXX);
  END;
  FOR YY := -1 DOWNT0 DESTS DO (* INITIAL ALL DESTROYER NODES *)
  BEGIN
    INITIALDES(DESSCY);
  END;
  (* INSERT STOP TIME FOR SIMULATION RUN IN EVENTQUEUE *)
  INSERTEVENTQUEUE(CALENDAR, MAXT, STOP, WW, NIL);
  WRITELN; WRITELN;
  WRITELN(' : BEGIN SIMULATION : ':51);
  WRITELN;
  LISTEVENTQUEUE;
  WRITELN; WRITELN;
  WRITELN('DO YOU WANT EVENTQUEUE LISTED DURING RUNNING THE PROGRAM ?');
  WRITE('TYPE Y<ES>, OR N<O> : ');
  READLN(SELECT1);
  WRITELN(SELECT1);
  WRITELN('DO YOU WANT EVENTS LISTED DURING RUNNING THE PROGRAM ?');
  WRITE('TYPE Y<ES>, OR N<O> : ');
  READLN(SELECT2);
  WRITELN(SELECT2);
  IF SELECT2 = 'Y' THEN
  REPORTHEADING;
  TIME := 0.0; (* INITIAL CLOCK FOR THE SYSTEM *)
  PACNO := 0; (* INITIAL VALUE OF PACKET NUMBER IN THE SYSTEM *)
  WRITELN;
  WRITELN('NOW...START THE SYSTEM SIMULATION NETWORK !');

```

```

WRITELN; WRITELN; WRITELN;
WRITE('T':4,'ORG,PAC#':14,'Tpac.':11,'NODE=>NODE':14,'Tcre.':12);
WRITE('Tnext':15,'Q-SIZE':13,'Tleave':12,'Q-SIZE':14,'K#':10);
WRITELN('Ttran.':12);
WRITELN; WRITELN; WRITELN;
END; (* INITSYS *)

```

```
(* BEGINNING OF THE MAIN PROGRAM *)
```

```
(* DESCRIBING THE VARIABLES IN MAIN PROGRAM *)
```

```

(*)
MAXT      : MAXIMUM TIME FOR SIMULATION RUN
D         : POINTER TO FIRST EVENT IN EVENTQUEUE
NPVAL     : EVENT THAT TAKE OFF FROM EVENTQUEUE
TT        : TIME OF OCCURENCE EVENT
ACTION    : ACTION TYPE OF CURRENT EVENT
PARM      : SPECIFY NUMBER OF NODE IN THE SYSTEM
NPTR      : PACKET INVOLVED IN CURRENT EVENT
ORI       : SPECIFY ORIGINATOR NODE THAT INVOLVED IN CURRENT
           : EVENT
DELS      : SPECIFY DELAY NODE THAT INVOLVED IN CURRENT EVENT
DESS      : SPECIFY DESTROYER NODE THAT INVOLVED IN CURRENT
           : EVENT
NO3       : NUMBER OF EXIT FROM ORIGINATOR NODE
TIM       : NEXT CREATION TIME FOR NEW PACKET
TEM4      : LEAVING TIME OF THE PACKET FROM DELAY NODE IN CASE
           : DELAY QUEUE IS EMPTY
TEM6      : LEAVING TIME OF THE PACKET FROM DELAY NODE IN CASE
           : DELAY QUEUE IS NOT EMPTY
NO2       : NUMBER OF EXIT FROM DELAY NODE
TEM2      : LEAVING TIME OF THE PACKET FROM ANOTHER DELAY NODE
           : AFTER COMPUTE EXITPATH, QUEUE EMPTY
TEM3      : LEAVING TIME FROM ANOTHER DELAY NODE, QUEUE
           : NOT EMPTY *)

```

```
BEGIN
```

```

WRITELN; WRITELN;
WRITELN('WELCOME TO THE SIMULATION SYSTEM FOR COMMUNICATION NETWORK');
WRITELN;
REPEAT
  REPEAT
    WRITELN('PLEASE ENTER MAXTIME...THE TOTAL SIMULATION TIME IN MINUTES');
    READLN(MAXT); (* READ MAXIMUM TIME FOR RUN SIMULATION PROGRAM *)
    WRITELN;
    WRITELN(MAXT:10:1);
    WRITELN; WRITELN;
    UNDERLINE;
    WRITELN;
    IF MAXT > 0.0 THEN
      GOODDATA := TRUE
    ELSE
      BEGIN
        GOODDATA := FALSE;
        WRITELN('*** YOU MADE AN ERROR IN THE INPUT,');
        WRITELN;
        WRITELN('*** MAXTIME MUST BE REAL AND NON-NEGATIVE NUMBER. ');
        WRITELN;
        WRITELN('*** PLEASE TRY AGAIN ***');
        WRITELN; WRITELN;
      END;
  END;
END;

```

```

UNTIL GOODDATA;
OPENFILE;
READDATA;
INITSYS;
REPEAT
  (* NOW WE RUN THE SIMULATOR FOR AS LONG AS THE USER SPECIFIED *)
  IF EVEQ.EFRONT^.TIME <= MAXT THEN
    BEGIN (* CURRENT TIME IN EVENTQUEUE < MAXIMUM TIME *)
      GETNEXTEVENT(NPVAL,TT,ACTION,PARM,NPTR);
      D := EVEQ.EFRONT;
      (* ADVANCE CLOCK TO NEXT EVENT IN EVENTQUEUE *)
      EVEQ.EFRONT^.TIME := D^.TIME;
      (* NOW SIMPLY CALL THE APPROPRIATE ROUTINE FOR THIS EVENT TYPE *)
      CASE ACTION OF
        CREATE : CREATEPACKET(ORI,PARM,TIM,NO3,TEM4,DELS,DESS,TEM6);
        LEAVE : LEAVEDELAY(NPTR,PARM,DELS,DESS,NO2,TEM2,TEM3);
        STOP : STOPSIM;
      END; (* OF CASE *)
      IF SELECT2 = 'Y' THEN (* REPORT ACTION AFTER PERFORMED EVENT *)
        REPORT(ACTION);
      IF SELECT1 = 'Y' THEN (* SHOW EVENTS IN EVENTQUEUE *)
        LISTEVENTQUEUE;
      END;
      UNTIL ACTION = STOP;
      WRITELN;
      LISTEVENTQUEUE;
      FINALREPORT(ORI,DELS,DESS); (* PRINT RESULTS OF SIMULATION RUN *)
      WRITELN('DO YOU WISH TO RUN THE SIMULATION AGAIN ...');
      WRITELN;
      WRITELN('FOR A DIFFERENT TIME VALUE OR DIFFERENT INPUT FILE ?');
      WRITELN;
      WRITE('TYPE Y<ES>, OR N<O> : ');
      READLN(SELECT);
      WRITELN(SELECT);
      DONE := (SELECT = 'N');
      CLOSE(INFILE);
      CLOSE(OUTFILE);
    UNTIL DONE;
    WRITELN; WRITELN;
    WRITELN('EXITING FROM THE SIMULATION PROGRAM...,THANK YOU');
    WRITELN;
  END.
  (*END OF SIMULATION PROGRAM*)

```

CHAPTER 9

INPUT AND OUTPUT FORMATS

CHAPTER 9

INPUT AND OUTPUT FORMATS

The successful development of simulation model have to have two processes to meet problem solving requirements. They are

1. Data Acquisition. The identification, specification, and collection of data.
2. Formulating and Printing Output.

In our simulation network program, we have the routines that handle two processes for input and output data of the simulation. The routine Readdata is defined for input data and we have five routines for the output data. There are Outputdata, Reportheading, Report, Listeventqueue, and Finalreport. The details for these routines are described as follows :

9.1 Input Format

The routine that handle the processes for read an input data of the simulation program is Procedure Readdata.

Input File Format

The input data file is a sequence of lines, each line .

contains only one datum. The input data file is broken down into two parts. First, is the data for originator nodes and second, is the data for delay nodes.

The first line of an input data file defines the number of originator nodes. The following lines describe the data for each originator node. Each originator node is described by three lines of data. The first line of data for each originator node gives the exit number for a packet to transfer from the originator node to another node in the system. The second line is Minwait. The third line of data for an originator node is Maxwait. The data for originator nodes continues line by line to the last originator node. This completes the first part of the input data file.

After the last line of data for originator nodes, the next line gives the number of delay node in the system. The next line gives the number of exits from each delay node. All delay nodes will have the same number of exits, some of these exits may have probability 0. The following lines describe the data for each delay node. In our example, each delay node is described by eight lines of data. The first line of data for each delay node gives the exit numbers, one exit per line. In our example, we have three lines for number of exits. The next lines define the probabilities for each respective exit, also one probability per line. In our

example, we have three lines for the probabilities. The next line defines minimum service time for a packet in this delay node (Holdmin). The following line defines the maximum service time for a packet in this delay node (Holdmax). The data for delay nodes continues line by line to the last delay node. This completes the second part of the input data file.

For example, we create an input data file for a network system with 2 originator nodes, 2 delay nodes, 5 destroyer nodes, and 3 exits from each delay node. With specific data, the input data file is shown in the following diagram.

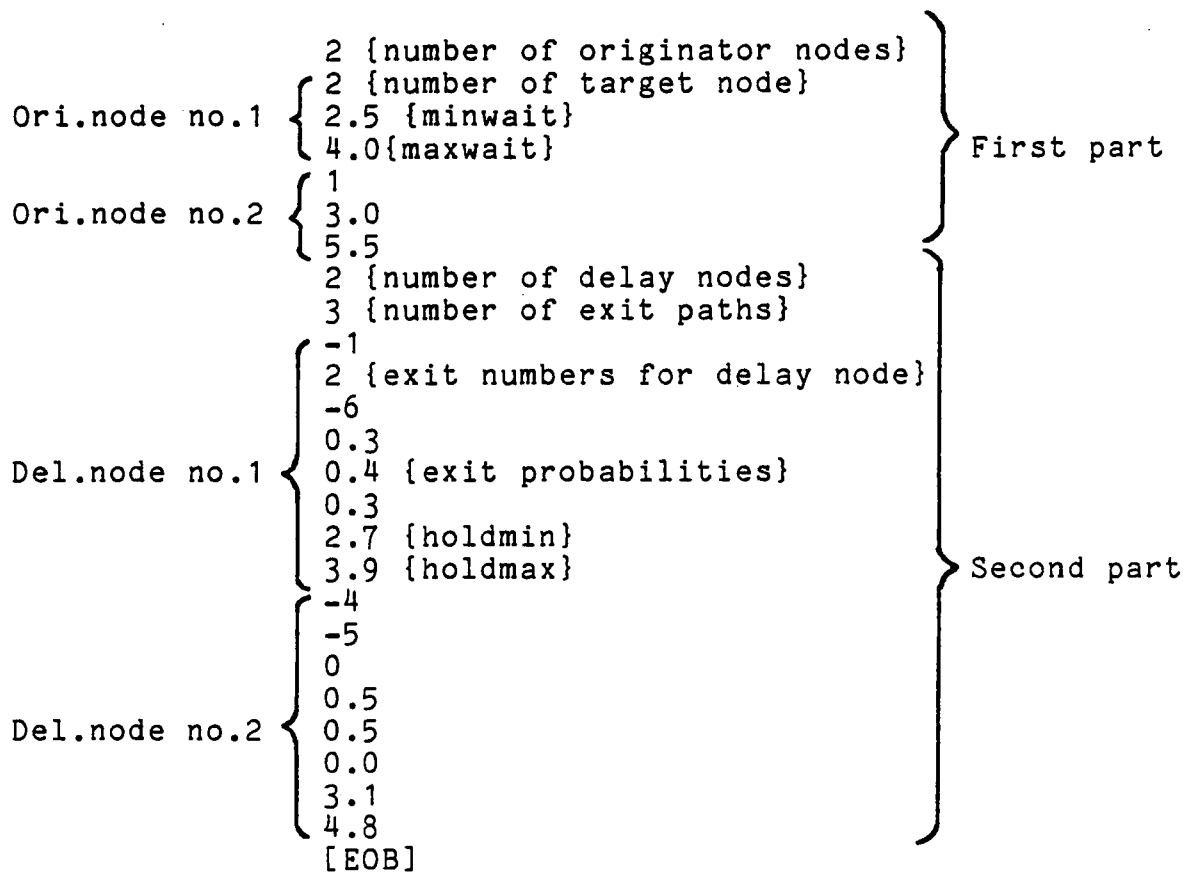


Figure 9.1 Format of input data file

Note: Non numeric data are comments and will not appear in an actual data file.

If the number of exits from a particular delay node is less than the number that we define, in this case 3, we use 0 represent the exit number and 0.0 for exitprob for each exit that is missed.

9.2 Output Formats

The simulation network program has 5 routines for print the output. The details for these output are described as follows :

Statistical Summaries for Simulation Network

The statistical summaries table of the simulation network system for originator node and delay node are constructed by Procedure Outputdata. The data in this table has been read from an input data file. From the previous example if we have 2 originator nodes, 2 delay nodes and 3 exits from each delay node, the statistical table for the simulation network system should be as follow :

Table Describing the Network System

For originator node

Originator node no.	Exit	Minwait	Maxwait
1	2	2.5	4.0
2	1	3.0	5.5

For delay node

Delay node no.	Exits	Exprob	Holdmin	Holdmax
1	-1 2 6	0.3 0.4 0.3	2.7	3.9
2	-4-5 0	0.5 0.5 0.0	3.1	4.8

Table 9.1 Statistical table for simulation network

Debug Output for Action Listing

The heading for the debug report and reports after the occurrence of an event are constructed by 2 procedures, Reportheading and Report. We have 6 types of actions that will be reported.

1. Action create and send packet to delay node, immediate servic.
2. Action create and send packet to delay node, put in delay queue.
3. Action create and send packet to destroyer node.
4. Action leave from delay node to delay node, immediate service.

5. Action leave from delay node to delay node, put in delay queue.
6. Action leave from delay node to destroyer node.

We give a sample debug output for action listing, using the above data, interspersed with our comments. This sample is shown in Table 9.2

In case of action = create and packet is sent to delay serve. We use the symbol 0 => DS represent in this case.

Time	Packet no.	Packet time in system	Event descriptor	Next create	Time leaving
8.4	1,3	8.4	0 1=>DS 2	11.3	9.8
9.7	3,5	9.7	0 2=>DS 1	12.0	11.6

In case of action = create and packet is sent to delay queue. We use the symbol 0 => DQ represent in this case.

Time	Pac. no.	Pac.time in system	Event descriptor	next create	time leaving	Q-size to node
10.5	2,4	10.5	0 2=>DQ 7	12.9	14.1	2
13.9	3,7	13.9	0 3=>DQ 1	15.0	15.9	1

In case of action = create and packet is sent to destroyer node.

We use the symbol 0 => K represent in this case.

Time	Pac. no.	Pac.time in system	Event descriptor	Next create	Total killed	Transit time
12.9	1,6	12.9	0 1=>K 3	14.8	3	0.0

Table 9.2 Sample of Debug Output for Action Listing

In case of action leave from delay to delay serve.
 We use the symbol D => DS represent in this case.

Time	Pac. no.	Pac.time in sys.	Event descriptor	Next Ev fromnode	Q-size fromnode	Time leaving
20.0	1,7	14.8	D 7=>DS 5	E	0	26.6
24.7	2,6	18.3	D 2=>DS 3	28.4	1	27.2

In case of action leave from delay to delay queue.
 We use the symbol D => DQ represent in this case.

Time	Pac. no.	Pac.time in sys.	Event descrp.	Next Ev fromnode	Q-size fromnode	Time leaving	Q-size tonode
25.9	3,8	20.7	D 1=>DQ 2	E	0	29.8	1
30.8	1,8	18.2	D 2=>DQ 4	32.6	1	34.6	2

In case of action leave from delay to destroyer node.
 We use the symbol D => K represent in this case.

Time	Pac. no.	Pac.time in sys.	Event descrp.	Next Ev fromnode	Q-size fromnode	Total killed	Transit time
31.5	2,7	27.9	D 7=>K 9	34.0	2	5	3.8
34.1	3,8	20.7	D 2=>K 1	E	0	8	5.9

Table 9.2 (cont.)

Sample of Debug Output for Action Listing

We put character 'E' in the column 'next event from node' if the delay queue of that node is empty. If the delay queue is not empty, we report the time of the next leave event from that node in this column.

Debug Output for Event Queue List

All of the events in the event queue are listed by the Procedure `Listeventqueue`. The number of events in the event queue vary by Procedure `Inserteventqueue` and Procedure `Getnextevent`. An example of the event queue lists is shown in Table 9.3

Schedule Time in Eventqueue	Action Type	Node Type	No
4.1	Create	Originator	1
5.8	Leave	Delay	2
6.0	Leave	Delay	1
7.4	Create	Originator	2
9.2	Create	Originator	3
9.8	Leave	Delay	7
.	.	.	.
.	.	.	.
.	.	.	.

Table 9.3 shown lists of the event in event queue

Summary Output

The summary report of the simulation network is produced by the Procedure `Finalreport`. The example of summary output is shown in Table 9.4

Originator node no.	Total packets created
1	5
2	3
⋮	⋮
⋮	⋮

Total packets created by originator nodes = 66

Delay node no.	Total packets in delay node	Average delay time at node
1	7	0.0
2	3	2.8
3	5	3.1
4	2	0.0
⋮	⋮	⋮
⋮	⋮	⋮

Total packets in delay nodes = 17
 Total delay time in delay nodes = 34.0
 Average delay time in delay node = 2.0

Destroyer node no.	Total packets destroyed	Average transit time
1	1	3.0
2	5	4.4
3	2	2.6
⋮	⋮	⋮
⋮	⋮	⋮

Total packet destroyed by destroyer nodes = 8
 Total lifetime of packets = 88.0
 Average transit time for packet = 11.0

Table 9.4 Summary Report of the Simulation Network

CHAPTER 10

ANALYSIS RESULTS OF SIMULATION NETWORK PROGRAM

CHAPTER 10

ANALYSIS RESULTS OF SIMULATION NETWORK PROGRAM

The program was coded and run on a VAX 11/780. In this chapter we will present the results of a simulation run for one system. Table 10.1 shows the data describing a simulation network system. These data were in an input data file, as described in Chapter 9. For this example, is used this input data file and ran the simulation for 100 time units (maxtime = 100.00). The initial value of the random number seed was zero. The summary results are presented in Table 10.2.

Originator node no.	Exit	Minwait	Maxwait
1	2	3.2	5.7
2	7	3.5	6.2
3	1	3.0	6.5

Table 10.1 Data describing simulation network system

Delay node no.	Exits	Exprob	Holdmin	Holdmax
1	-1 2 6 -10 7	.3 .1 .2 .3 .1	2.0	3.5
2	4 1 -3 -8 0	.2 .1 .4 .3 .0	1.7	2.9
3	-4 5 6 -6 0	.4 .3 .2 .1 .0	2.6	3.8
4	6 -5 -9 5 3	.4 .2 .1 .1 .2	3.0	6.0
5	2 -4 4 -2 -7	.3 .2 .1 .2 .2	2.5	5.0
6	3 7 -1 0 0	.3 .4 .3 .0 .0	3.1	4.5
7	-1 6 -7 -10 3	.1 .1 .4 .2 .2	2.2	3.9

Table 10.1(Cont.) Data describing simulation network system

From the above table, this network model contains 3 originator nodes, 7 delay nodes, and 10 destroyer nodes. Each delay node has 5 exits or less than 5, to transfer packets to another node. We now present the results of a simulation run.

Results of a simulation run

Simulation for 100 time units

Originator node no.	Total packets created
1	24
2	21
3	23

Total packets created by originator node = 68

Delay node no.	Total packets in delay node	Average delay time at node
1	24	0.053
2	30	0.428
3	6	0.003
4	6	0.000
5	2	1.368
6	9	0.223
7	26	1.313

Total packets in delay nodes = 103
 Total delay time in delay nodes = 52.988
 Average delay time in delay node = 0.514

Destroyer node no.	Total packets destroyed	Average transit time
1	14	5.415
2	1	22.978
3	9	4.018
4	3	9.998
5	1	10.716
6	1	8.471
7	11	5.041
8	13	2.665
9	1	8.115
10	11	4.865

Total packets destroyed by destroyer nodes = 65
 Total lifetime of packets = 335.858
 Average transit time for packet = 5.167
 CPU seconds used = 0.56

Table 10.2 Results of the simulation network model

With 100 units running time, originator nodes number 1, 2 and 3 created 24, 21 and 23 packets, respectively. The total of packets created in this network model is 68. Now, consider closely the results for delay nodes. From table 10.2 we have an average delay time of packets in each delay node. Delay node number 4 has an average delay time 0.000, this means no delay time in this node. Note that delay time does not include service time. So we can improve system performance by adjusting the parameter values in exitlists, in order to transfer packets to delay node number 4. Also, delay node numbers 5 and 7 have long delay times because of the value of parameters Holdmin, Holdmax for those nodes. To minimize delay time for each delay node, we can adjust the parameter values of Holdmin and Holdmax by studying all of the information that relates to these values in order to decrease waiting time in the delay queue. By adjusting these parameter values until we have a proper value of Holdmin and Holdmax for each delay node we can decrease the value 0.514, average delay time for the packet in delay node.

Again, from table 10.2 we also have average transit time for packet. In this case the average transit time is 5.167, this means in our network model, a packet take time 5.167 between when it is created until it is destroyed. To minimize this time value, we can adjust the parameter values of input accordingly.

For example, if we now adjust the parameter values of input as show in Table 10.3. The new results of a simulation run are presented in Table 10.4.

Originator node no.	Exit	Minwait	Maxwait
1	2	3.2	5.7
2	7	3.5	6.2
3	1	3.0	6.5

Delay node no.	Exits	Exprob	Holdmin	Holdmax
1	-1 2 6 -10 4	.3 .1 .2 .3 .1	2.0	3.5
2	4 1 -3 -8 0	.2 .1 .4 .3 .0	1.7	2.9
3	-4 5 6 -6 0	.4 .3 .2 .1 .0	2.6	3.8
4	6 -5 -9 5 3	.4 .2 .1 .1 .2	1.5	2.6
5	2 -4 4 -2 -7	.3 .2 .1 .2 .2	2.1	3.3
6	3 4 -1 0 0	.5 .2 .3 .0 .0	3.1	4.5
7	-1 6 -7 -10 3	.1 .1 .4 .2 .2	1.9	2.8

Table 10.3 Data describing simulation network system

Results of a simulation run

Simulation for 100 time units

Originator node no.	Total packets created
1	23
2	21
3	22

Total packets created by originator node = 66

Delay node no.	Total packets in delay node	Average delay time at node
1	28	0.617
2	28	0.220
3	14	0.056
4	7	0.132
5	3	0.000
6	16	1.949
7	21	0.000

Total packets in delay nodes = 117
 Total delay time in delay nodes = 56.332
 Average delay time in delay node = 0.481

Destroyer node no.	Total packets destroyed	Average transit time
1	10	6.618
2	0	0.000
3	8	3.499
4	8	10.624
5	3	6.519
6	0	0.000
7	9	6.127
8	10	2.993
9	2	8.461
10	12	3.367

Total packets destroyed by destroyer nodes = 62
 Total life time of packets = 341.121
 Average transit time for packet = 5.502
 CPU seconds used = 0.56

Table 10.4 Results of the simulation network model

From Table 10.4, after we adjust some parameter values, the value of average delay time decreases from 0.514 to 0.481. Also delay times in delay node numbers 5 and 7 are decreased. But the average transit time for a packet is increased from 5.167 to 5.502. The reason is, we did not adjust the parameters Holdmin and Holdmax of every delay node, we just adjusted those values for delay node numbers 4, 5, and 7. Therefore, the simulation program must be able to run frequently, with a different set of parameters in order to find optimal parameters.

Most of the parameter values of input data for network modeling have an influence on the average transit time for packet. If we find a proper value of service time in a delay node by adjusting the parameters Holdmin and Holdmax then the average transit time should be decreased. The reason for the previous statement is logically, if the service time is short, then delay time is short and if delay time and service time are short, then total transit time is short. The other factors that may be adjusted in order to improve average transit time are

(1) Number of delay nodes and number of exits for each delay node or Work distribution. If we have many delay nodes in the network system, we can distribute the packets to those delay nodes. So, delay node will not be busy and the packets

do not wait in delay queues. Then the average transit time is decreased. And also if we have a lot of exits for each delay node, it produce the same result that is we can decrease the average transit time for packet.

(2) The value of Minwait and Maxwait for each originator node or Decreasing the load on the system. If we decrease these values, delay node will not be as busy transferring packets through the system, because before the next packet is created, the former packet is already transfered from delay node to another node. When delay nodes are not as busy, average transit time is decreased.

Because simulation is a design tool, we can improve performance of the system by adjusting the parameter values. These values can be changed to minimize the time values for the network model.

APPENDIX A

FLOW CHART OF FUNCTIONS MANAGING QUEUE

APPENDIX A
 FLOW CHART OF FUNCTIONS MANAGING QUEUE

INSERTEVENTQUEUE

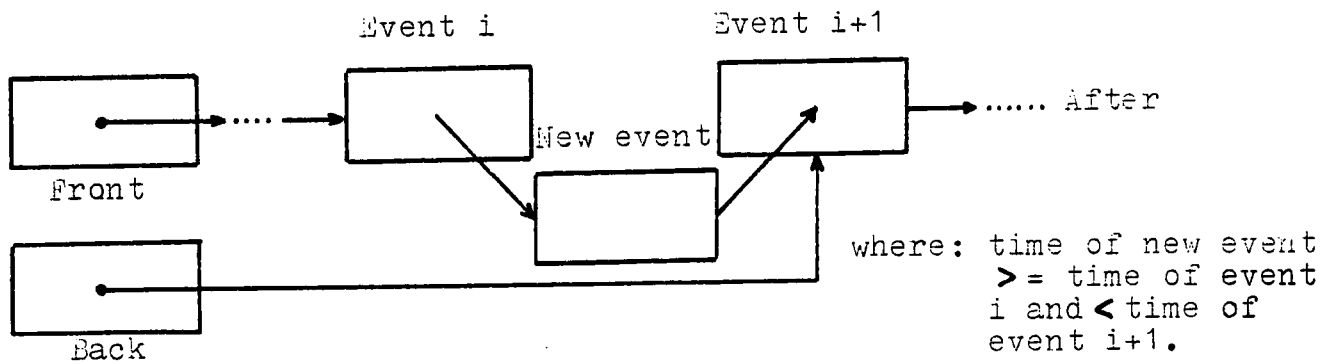
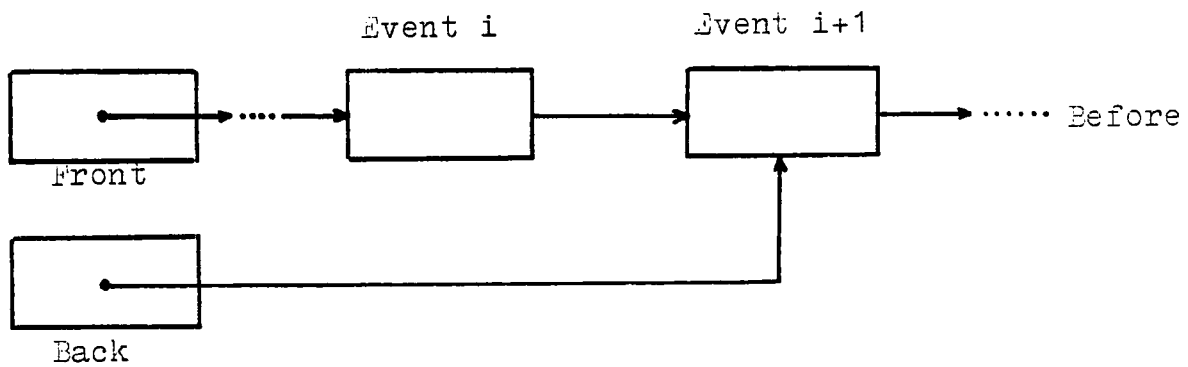


Figure shows flow chart of procedure Inserteventqueue

GETNEXTTEVENT

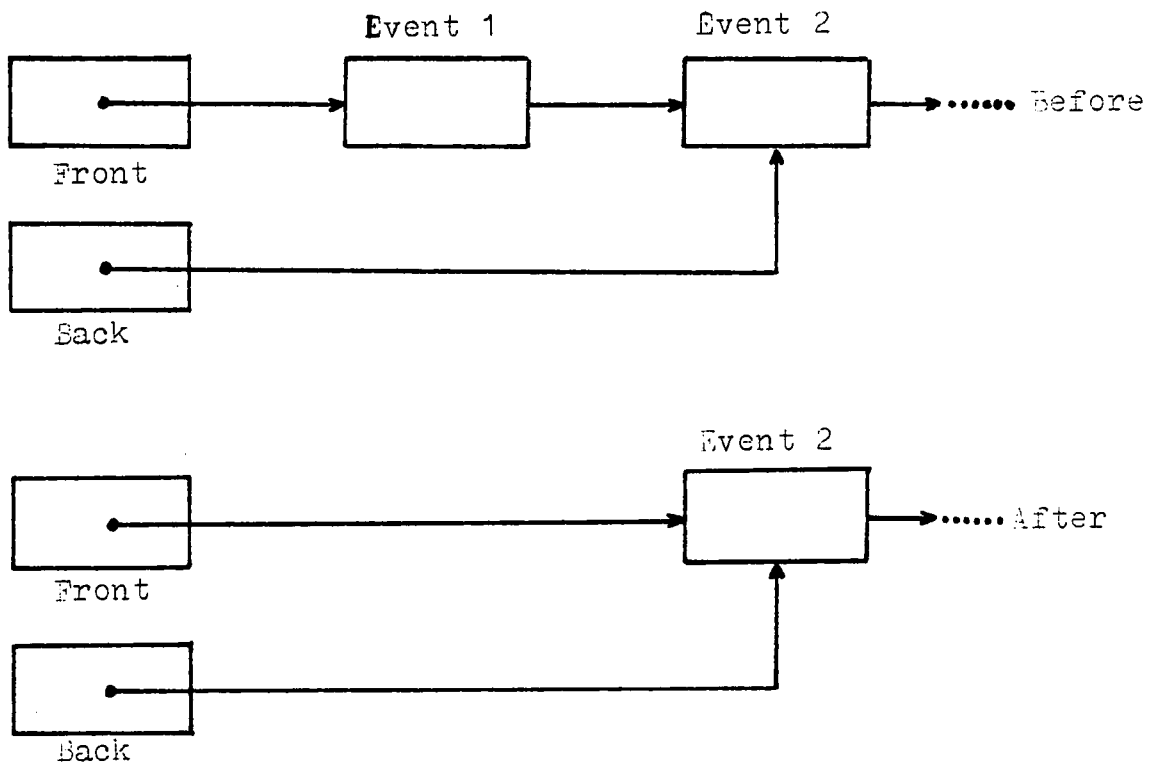


Figure shows flow chart of procedure Getnexttevent

INSERT

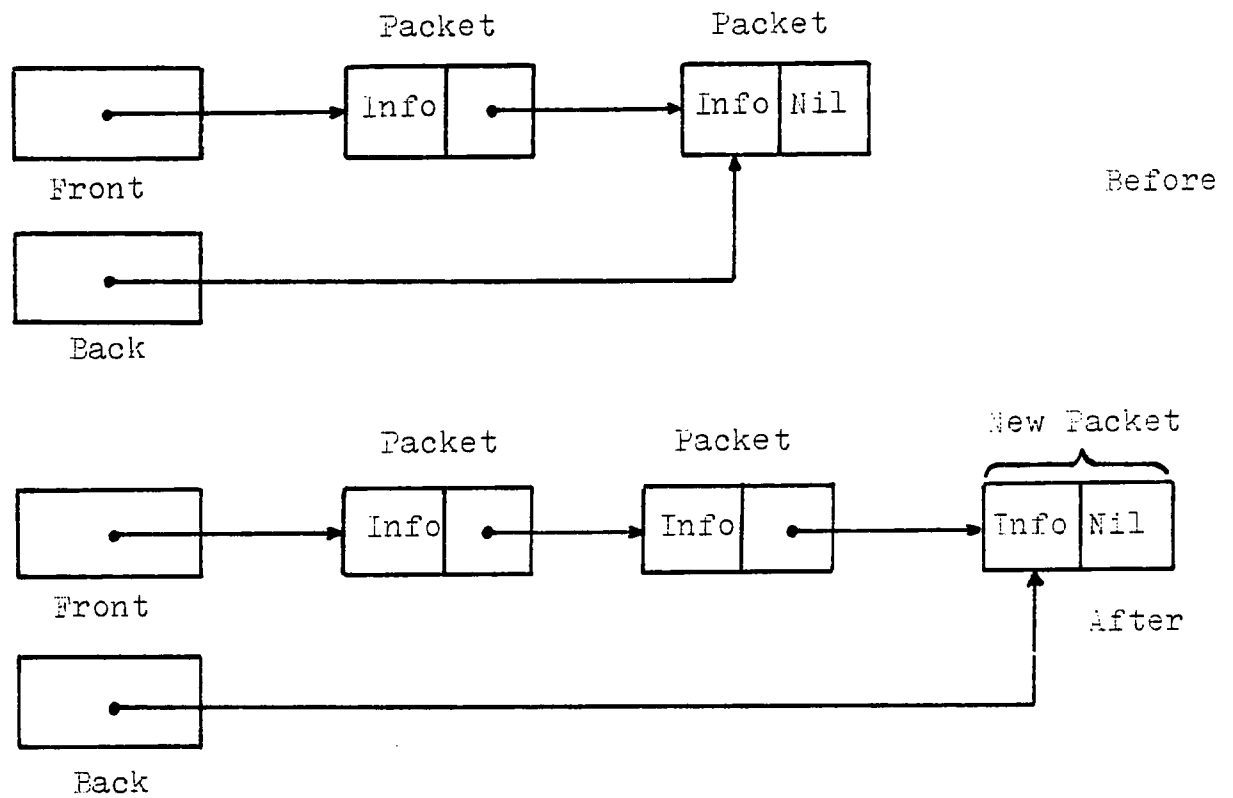


Figure shows flow chart of procedure Insert

DELETE

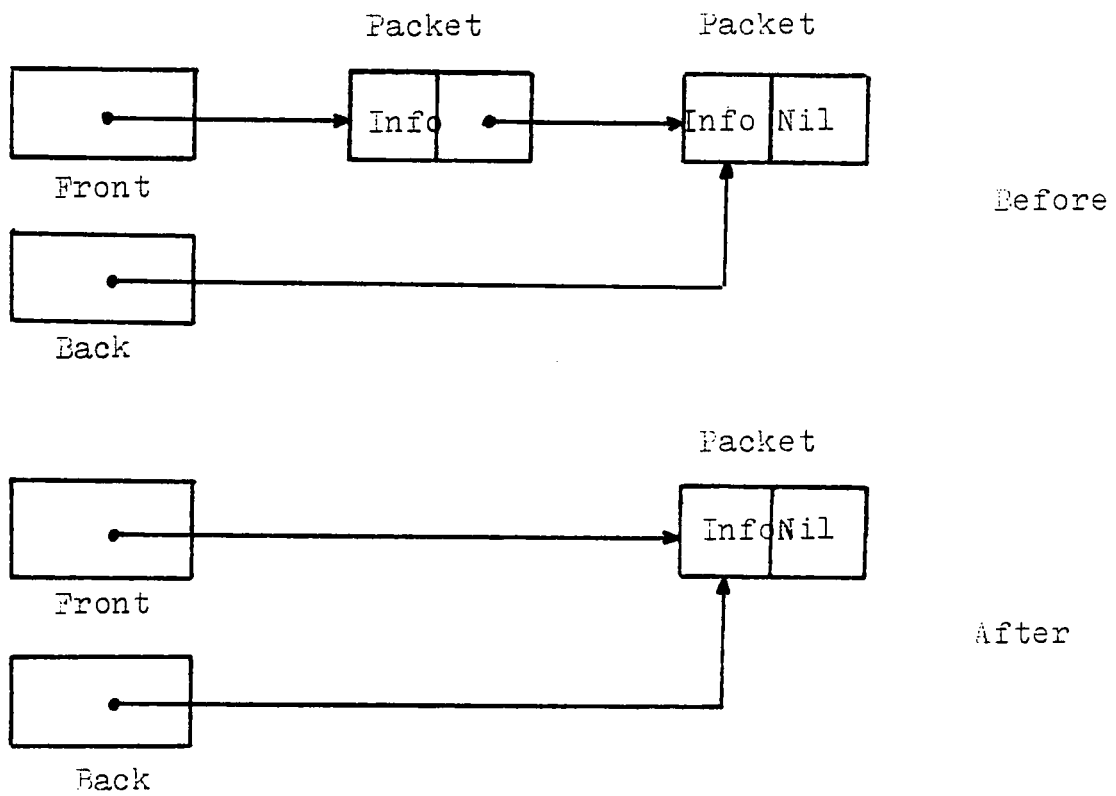


Figure shows flow chart of procedure Delete

APPENDIX B
TESTING RANDOM NUMBER

APPENDIX B
TESTING RANDOM NUMBER

Our simulation network system has a function generating pseudo random number to perform the model. In our modeling we use the following formula to generate a sequence of random numbers on the interval [0,1].

$$\begin{aligned} \text{Seed} &= (\text{multiplier} \times \text{Seed} + \text{increment}) \text{ mod modulus} \\ \text{Random number} &= \text{Seed}/65536 \end{aligned}$$

where

$$\begin{aligned} \text{modulus} &= 65536 \\ \text{multiplier} &= 25173 \\ \text{increment} &= 13849 \\ \text{seed} &= \text{integer number} \end{aligned}$$

This formula can produce different sequence of 65,536 real random numbers by altering the initial parameter, seed. For example, if we pick 2 different numbers of initial seed, say 0 and 4 and we require 100 real random numbers, we will get different sequences. The two different sequences are shown in the following Tables.

0.211	0.744	0.476	0.262	0.212	0.936	0.793	0.474	0.480	0.290
0.305	0.871	0.936	0.163	0.560	0.937	0.753	0.167	0.356	0.406
0.352	0.160	0.739	0.263	0.405	0.252	0.005	0.742	0.834	0.802
0.702	0.328	0.810	0.419	0.626	0.754	0.757	0.963	0.825	0.005
0.345	0.543	0.595	0.316	0.497	0.518	0.484	0.171	0.383	0.499
0.287	0.304	0.428	0.344	0.053	0.700	0.301	0.162	0.077	0.593
0.925	0.313	0.906	0.468	0.472	0.407	0.339	0.058	0.364	0.302
0.420	0.349	0.272	0.109	0.448	0.574	0.119	0.186	0.970	0.218
0.076	0.143	0.781	0.015	0.566	0.839	0.929	0.950	0.259	0.384
0.711	0.257	0.079	0.137	0.674	0.420	0.196	0.707	0.614	0.175

Table of sequence 100 real random numbers, seed = 0

0.748	0.497	0.803	0.960	0.600	0.489	0.603	0.762	0.421	0.408
0.677	0.388	0.884	0.363	0.322	0.074	0.911	0.714	0.085	0.304
0.673	0.069	0.764	0.812	0.592	0.289	0.404	0.583	0.340	0.482
0.303	0.851	0.839	0.008	0.008	0.101	0.261	0.477	0.315	0.066
0.028	0.748	0.272	0.731	0.811	0.929	0.175	0.331	0.533	0.381
0.571	0.350	0.365	0.713	0.257	0.523	0.730	0.784	0.281	0.833
0.298	0.704	0.937	0.515	0.993	0.832	0.775	0.053	0.958	0.777
0.590	0.183	0.697	0.399	0.430	0.887	0.801	0.808	0.591	0.401
0.218	0.361	0.620	0.209	0.121	0.670	0.314	0.046	0.931	0.593
0.720	0.895	0.319	0.239	0.134	0.991	0.209	0.968	0.105	0.818

Table of sequence 100 real random numbers, seed = 4

The program also has a function managing simulation time. The times involved are creation time for a new packet and service time of a packet in a delay node. Both of these times can be called 'event times'. The event times are always updated frequently during simulation run. We use the function shows below to compute event times in our simulation

network model.

$$\text{Event time} = \text{Random number} * (\text{High} - \text{Low}) + \text{Low} + \text{Current time}$$

This formula uses the sequence of random numbers on the interval [0,1] that we are discussed earlier and two parameter values called Low and High (real numbers). These two parameters are defined in the input data file before running the simulation, and are differed for different event types in the model.

Suppose, we defined the value of parameter Low and High as 2.5 and 4.5 respectively. Then we use the sequence of real numbers generated above to compute the event time in the model. A list of 100 event times are shown in the following table

2.923	3.988	3.453	3.024	2.925	4.372	4.085	3.449	3.406	3.079
3.110	4.243	4.373	2.827	3.621	4.375	4.006	2.835	3.212	3.313
3.204	2.821	3.977	3.027	3.310	3.004	2.510	3.985	4.167	4.104
3.904	3.155	4.121	3.338	3.753	4.008	4.014	4.426	4.151	2.510
3.190	3.586	3.691	3.132	3.493	3.537	3.468	2.842	3.267	3.498
3.074	3.109	3.355	3.187	2.605	3.901	3.102	2.824	2.653	3.687
4.350	3.126	4.313	3.436	3.444	3.314	3.177	2.616	3.228	3.105
3.341	3.197	3.044	2.718	3.397	3.647	2.738	2.872	4.439	2.935
2.653	2.786	4.062	2.530	3.932	4.178	4.358	4.400	3.019	3.268
3.922	3.014	2.658	2.744	3.847	3.341	2.892	3.914	3.727	2.849

Event times, using initial seed = 0, Low = 2.5, High = 4.5

The values in previous Table use the initial value 0 for generate random numbers. If we take initial value 4 for

generate random numbers, it means that the list of event times will be changed. We can compare those list of numbers from the next Table.

3.996	3.494	4.107	4.421	3.699	3.479	3.706	4.023	3.342	3.316
3.854	3.276	4.268	3.277	3.144	2.648	4.321	3.928	2.671	3.108
3.845	2.638	4.028	4.124	3.683	3.078	3.308	3.666	3.180	3.464
3.106	4.202	4.178	2.517	2.517	2.701	3.022	3.454	3.131	2.632
2.556	3.995	3.044	3.963	4.123	4.358	2.850	3.163	3.566	3.263
3.642	3.201	3.231	3.926	3.015	3.545	3.960	4.069	3.061	4.165
3.097	3.909	4.375	3.529	4.486	4.164	4.050	2.606	4.471	4.055
3.680	2.865	3.895	3.299	3.359	4.274	4.102	4.115	3.681	3.302
2.936	3.222	3.739	2.918	2.741	3.840	3.127	2.591	4.361	3.685
3.939	4.289	3.139	2.978	2.769	4.482	2.918	4.437	2.709	4.137

Event times using initial seed = 4, Low = 2.5, High = 4.5

Finally, the conclusion about the random numbers that are used in the simulation network model is, if we select a different initial seed, we obtain a different sequence of random numbers. That causes a change of event times in the model. When the event times are different, the results of the simulation run for the network system are different too. An example of the different results are shown in the following two Tables.

Results of the Simulation Run for Network System

Originator node no.	Total packet created
1	24
2	21
3	23

Total packet created by originator nodes = 68

Delay node no.	Total packets in delay node	Average delay time at node
1	24	0.053
2	30	0.428
3	6	0.003
4	6	0.000
5	2	1.368
6	9	0.223
7	26	1.313

Total packets in delay nodes = 103
 Total delay time in delay nodes = 52.988
 Average delay time in delay node = 0.514

Destroyer node no.	Total packets destroyed	Average transit time
1	14	5.415
2	1	22.978
3	9	4.018
4	3	9.998
5	1	10.716
6	1	8.471
7	11	5.041
8	13	2.665
9	1	8.115
10	11	4.865

Total packet destroyed by destroyer nodes = 65
 Total life time of destroyer nodes = 335.858
 Average transit time for packet = 5.167

Table: Shows the results of simulation network used initial value 0 for the first random number and 100 units for the maximum simulation run time

Results of the Simulation Run for Network System

Originator node no.	Total packets created
1	23
2	21
3	21

Total packets created by originator nodes = 65

Delay node no.	Total packets in delay node	Average delay time at node
1	24	0.313
2	27	0.139
3	7	0.310
4	6	0.000
5	3	0.000
6	6	0.905
7	25	0.658

Total packets in delay nodes = 98

Total delay time in delay nodes = 35.288

Average delay time in delay node = 0.360

Destroyer node no.	Total packets destroyed	Average transit time
1	15	5.501
2	0	0.000
3	7	3.636
4	3	9.974
5	0	0.000
6	0	0.000
7	11	7.081
8	12	2.493
9	1	6.919
10	13	3.818

Total packet destroyed by destroyer nodes = 62

Total lifetime of destroyer nodes = 302.251

Average transit time for packet = 4.875

Table: Shows the results of simulation network used initial value 4 for the first random number and 100 units for the maximum simulation run time

Because of the different values of the seed, different results occurred. From the above Tables, the number of packets involved in the system, average delay time and average transit time for packets are quite different.

Because generating random numbers is an important function for the simulation. The user should be careful, before running this simulation network program, to choose carefully the initial value of the seed.

APPENDIX C

GLOSSARY

APPENDIX C

GLOSSARY

ACTIVITY: a process that cause change in the system.

ATTRIBUTE: a property of an entity.

CLOCK: current event pointer of event in event queue.

COMMUNICATION NETWORKS: the system that transfer and processing of discrete units of information within the network.

CREATE EVENT: event that create a new packet at originator node.

CREATION TIME: time for create new packet at originator node.

DELAY NODE: a location in the network where packets wait for service.

DELAY QUEUE: queue of delay node that handle packets before send them to other nodes in system.

DESTROYER NODE: node that use to destroy or delete packets from network.

DISCRETE SIMULATION: model that reproduce the activities and learn something about the behavior and performance potential of the system.

ENTITY: an object of interest in a system.

EVENT: a combination of activities that occur in response to some single activity.

EVENT QUEUE: a linked list data structure that contains a list of all the events that occur in the network system.

EXIT: a target node number of exit path.

EXITS: list of exit paths to other nodes in the system.

EXIT PATH: possible flows of packet from node to node.

EXPROB: probabilities for exit paths of delay nodes.

HOLDMAX: maximum service time in delay node.

HOLDMIN: minimum service time in delay node.

LEAVING TIME: time that packet leave from delay node.

LEAVE EVENT: event that packet leave from delay node going to another node in the system.

LINK: the path that connect between two nodes.

MINWAIT: minimum time until next packet creation.

MAXWAIT: maximum time until next packet creation.

MODEL: a representation of a system to describe a detail and prediction the behavior of the system.

NETWORK: the system that two or more elements interact with one another.

NETWORK MODELING: the model consists of an interrelated set of nodes and links.

NODE: a point in a network which is a junction of lines.

ORIGINATOR NODE: node that create packets and routes them into the network system.

PACKET: an entity that contain information of interest and flow through the network.

QUEUE: a data structure used for dynamic temporary storage of elements.

RANDOM NUMBERS: numbers that generate from the algorithm for which contain the value of multiplier, increment, and modulus.

SIMULATION: the process of conducting experiments on the

computer model of a dynamic system.

STOP EVENT: event that stop simulation program.

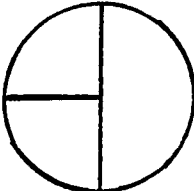
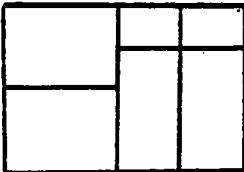
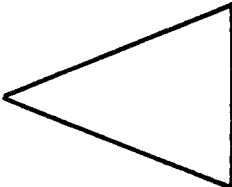

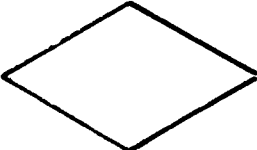
SYSTEM: an aggregation or assemblage of objects joined in some regular interaction or interdependence.

TRANSIT TIME: time that each packet takes since creation until destruction.

APPENDIX D

SYMBOL REPRESENTATION

APPENDIX D
SYMBOL REPRESENTATION

SYMBOL	REPRESENTS
	ORIGINATOR NODE: node that create packets and send them through the system.
	DELAY NODE: a single server queue.
	DESTROYER NODE: node that receives a packet and removes it from the system.
	PROCESSING: a group of program instructions which perform a processing function of the program.
	DECISION: the decision function used to document points in the program where a branch to alternate paths is

possible.



TERMINAL: the beginning, end, or a point of interruption in a program.

BIBLIOGRAPHIES

BIBLIOGRAPHIES

- Alan, A.; Pritsker, B.; and Pegden, Claude Dennis.
Introduction to Simulation and SLAM. New York: A
Halsted Press Book, John Wiley and Sons, 1979.
- Bobillies, P.A.; Kahan, B.C.; and Probst, A.R. Simulation
with GPSS and GPSS V. New Jersey: Englewood Cliffs,
1976.
- Bulgren, William G. Discrete System Simulation. Englewood
Cliffs, New Jersey: Prentice-Hall Inc., 1982.
- Cellier, Francois E. Progress in Modeling and Simulation.
New York: Academic Press, 1982.
- Cross, M.; Gibson, R.D.; O'Carroll M.J.; and Wilkinson
T.S. Modeling and Simulation in Prastic. New York:
John Wiley and Sons Inc., 1978.
- Goodman, S.E., and Hedetniemi, S.T. Introduction to the
Design and Analysis of Algorithms. University of
Verginia: Mc Graw-Hall, Inc., 1977.
- Grogono, Peter. Programming in PASCAL. Concordia
University, Montreal: Addison-Wesley, Publishing
Company, Inc., 1980.

Law, Averill M., and Kelton, David W. Simulation Modeling and Analysis. New York: Mc Graw-Hill Inc., 1982.

Lehman, Richard S. Computer Simulation and Modeling. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 1977.

Maisel, Herbert, and Gnugnoli, Giuliano. Simulation of Discrete Stochastic Systems. Chicago: Science Research Associates Press, 1972.

Moore, Laurence J., and Clayton, Edward R. GERT Modeling and Simulation. New York: Mason/Charter Publishers, Inc., 1976.

Schneider, Michael G., and Bruell, Steven C. Advance Programming and Problem Solving with PASCAL. New York: John Wiley and Sons Inc., 1981.

Schoemaker, S. Computer Networks and Simulation. New York: North-Holland Publishing Company, 1978.

Sharma, Roshan Lal; De Sousa, Paulo J.T.; and Ingle, Ashok D. Network Systems. New York: Van Nostrand Reinhold Company, 1982.

Zaks, Rodney. Introduction to PASCAL Including UCSD PASCAL. University of California: Sybex Inc., 1981.

Zeigler, Bernard P. Theory of Modeling and Simulation. New
York: John Wiley and Sons Inc., 1976