COMPUTER SIMULATION MODEL FOR A CSMA/CD LOCAL AREA NETWORK UTILIZING A MULTIRATE VOICE CODING TECHNIQUE FOR LOAD CONTROL

Victor S. Frost
Edward M. Friedman

Telecommunications and Information Sciences Laboratory
University of Kansas
Lawrence, Kansas 66045

Technical Report TISL-6731-2
May 1985

Supported by

National Science Foundation
Presidential Young Investigator Award

and

AT&T Information Systems

Principal Investigator: Victor S. Frost

ABSTRACT

A simulation model developed at the University of Kansas Telecommunications and Information Sciences Laboratory models an Ethernet-like Local Area Network. This simulation model is used to predict the performance of a voice/data system. The SLAM simulation language is used to model the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) communications protocol. The software developed simulates a CSMA/CD network and is fully documented. The simulation model was developed so that networks with a combined loading of voice and data could be analyzed. Simulation results were collected to determine the feasibility of using a multirate voice coding system for load control. The voice coding rate will be decreased when the network traffic is high. The decrease in voice coding rate causes less voice packets to access the network and decreases the voice quality. The method of changing the voice coding rate is implemented as a feedback system which uses the rate of collisions per millisecond to obtain an indication of the amount of traffic on the network. The methods available for coding the voice signal at multiple rates are not discussed in detail.

The CSMA/CD simulation model has been validated, a comparison of the simulated results to Shoch's measurements shows that the model accurately predicts the performance of a real system. It is shown that the performance of CSMA/CD Local Area Networks, which have a combined loading of voice and data, can be improved by using multirate voice coding for load control. The simulation results of the multirate voice coding system show that the feedback system properly adjusts to changing load conditions and that the use of multirate voice coding for load control allows a considerable increase in the number of conversations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The desire to have a simulation model of the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) communications protocol, and the ability to perform simulation studies of voice/data networks prompted this work. The CSMA/CD protocol is perhaps the most widely used packet communication system for Local Area Networks. This is due to the extensive application of the Ethernet Local Area Network (LAN). The goal of this research is to develop a CSMA/CD simulation model which can be used for performance studies of Ethernet-like computer communication networks. The simulation model is flexible so that any network configuration can be studied. The specific study presented here gives the performance of a voice/data network. The voice/data network uses variable rate voice coding for load control.

This report is divided into three chapters. Chapter 2 explains the SLAM simulation language which was used to model the CSMA/CD network. Chapter 3 gives a description of the code developed which simulates the CSMA/CD network. Chapter 4 describes the voice/data simulation study.

The Simulation Language for Alternative Modeling (SLAM) was used as the basic tool for developing the CSMA/CD simulation model. SLAM is a multipurpose simulation language and is useful for modeling communication networks. SLAM provides the flexibility needed to reconfigure the CSMA/CD simulation model to any particular network. In Chapter 2, the simulation language SLAM is introduced in a general sense, with reference to communication network modeling. There are many aspects of SLAM which have not been used in the CSMA/CD simulation model, these aspects are not covered in Chapter 2. Chapter 2 has been included for completeness, the reader can understand the CSMA/CD simulation model by referring to this chapter without referencing the available literature on SLAM.

The CSMA/CD simulation model is fully documented in Chapter 3. This chapter has been included so that expansion or modification of the software can be done easily. Further research may involve performance studies of a particular network that has a different configuration, but uses the basic CSMA/CD protocol. With the aid of Chapter 3, the analyst can modify the code with a minimum amount of effort. The CSMA/CD simulation model was initially developed for voice/data simulation studies, but can be configured for performance studies of any Ethernet-like network.

The voice/data simulation study requires that the CSMA/CD model be modified. The original simulation model was for only data traffic. Voice packets require a certain amount of special attention. The voice packets are generally smaller than the data packets. The voice packet interarrival time is constant, whereas the data packet interarrival time is generally assumed to be exponential. There have been several previous studies [12, 13, 14] done to determine the performance of a LAN that has a combined loading of voice and data. None of the previous studies address the problem discussed in Chapter 4. The simulation study described in Chapter 4 has the voice coding rate changing dynamically according to the level of network traffic. By lowering the voice coding rate when the network traffic is high, the voice quality is traded for network load. If the voice coding rate is lowered, less voice packets are attempting to transmit over the network, and a larger number of data packets can be transmitted. When the traffic decreases, the voice coding rate will be increased which causes more voice packets to access the network. By lowering the voice coding rate when the traffic increases, the network is able to handle a greater amount of data traffic without a substantial increase in the data packet delay. In addition, more voice conversations can take place when the multirate voice coding is used. If there are few

conversations, the voice coding rate remains high, however, if the number of conversations increases the voice coding rate will be lowered so that a greater number of conversations can take place.

The ability of the same voice source to generate packets at different coding rates has been assumed. That is, the use of multirate voice coding techniques has been assumed in Chapter 4, however, voice coding is not the subject of this paper. Chapter 4 is a proof of concept study, which shows that the use of variable rate coding can improve the network performance of voice/data CSMA/CD packet communication networks.

2.0 SIMULATION WITH SLAM

In this chapter, the Simulation Language for Alternative Modeling (SLAM) will be introduced to the extent that the CSMA/CD simulation model can be understood. Therefore, this chapter will be necessarily brief. SLAM is a useful simulation tool, for a complete description of SLAM see [1]. This chapter is also intended to be a quick reference or tutorial covering the basic SLAM concepts.

SLAM is a multipurpose simulation tool which is convenient for modeling communication networks. In SLAM, there are three approaches to simulation. The first is Network Modeling. Here the user can represent a process and the flow of entities through the process. The representation is done using blocks or nodes which are provided by SLAM. The user essentially sets up a block diagram using the nodes provided by SLAM. The second approach is known as Discrete Event Modeling. It allows the user greater flexibility than the Network approach. The added flexibility comes from having greater control over the simulation. In Discrete Event Modeling the user models the changes in the state of a system which occur at discrete points in time. The third modeling approach is Continuous Modeling. This method allows the user to model systems which change continuously over time and can be described by a set of differential equations. The continuous Modeling approach was not used in the CSMA/CD simulation model, to be described in Chapter 3, and will not be discussed in this chapter.

In SLAM an event calendar is set up. This event calendar is a file that holds the specific events which are due to occur. Events are placed on the calendar in chronological order by SLAM, and at the proper time are executed. SLAM relieves the user from keeping track of the order in which execution is to take place. In a Network Model, SLAM automatically places events

on the event calendar. In a Discrete Event model, the user places events on the calendar and can remove events from the calendar.

In this chapter, the Network blocks used to model the CSMA/CD system will be described. The Discrete Event subroutines used in the CSMA/CD simulation model will also be described. The method of combining Network Modeling and Discrete Event Modeling will also be discussed. Chapter 3 will describe the CSMA/CD simulation model in detail.

2.1 Network Modeling

In the Network Modeling approach, the system to be simulated is put together using the network blocks provided by SLAM. The user is basically putting together a block diagram of the system to be simulated. This approach is very convenient for modeling queueing systems, such as the arrival of packets to a node on a Local Area Network, and the build up of packets at the node. Although the network blocks provided by SLAM can completely describe the system, SLAM requires that certain statements be included in the model. These statements are used to specify the number of attributes per entity, the number of files used, and the length of the simulation, among other things. The attributes are a list of variables which are used to describe certain characteristics of the entity.

In this section, the SLAM Network Modeling blocks used in the CSMA/CD simulation model will be described. That is, the specific SLAM blocks and SLAM statements used to model the CSMA/CD network will be described. In addition, the collection of statistics will be described.

2.1.1 SLAM Blocks

In this section, a brief description of the SLAM blocks used to model the CSMA/CD network will be given. The CREATE block is used to generate packets at a user specified rate. The structure of the CREATE block is as follows:

CREATE,TBC,TF,MA,MC,M;

where,

TBC = time between creations, can be constant or random

TF = time 1st entity is created

MA = mark time, places creation time in the MAth attribute

MC = maximum number of creations

M = number of branches

The value specified for TBC can be a constant or samples from a random distribution. In SLAM there are several distributions provided, they are listed in Table 7-1 of [1]. The semicolon is the last character on every line of code, SLAM ignores everything after the semicolon. As entities or packets in the case of the CSMA/CD network are generated using the CREATE block, an attribute array is given to each packet. The attribute array is essentially a group of variables which can be used to specify certain characteristics of the specific packet. The packet will carry the list of attributes as it travels through the network. SLAM provides for altering and setting the attributes associated with a particular entity, this is done using the ASSIGN block. The structure of the ASSIGN block is given below.

ASSIGN, var=value,var=value,...,M;

where,

M = number of branches the entity is to be routed through

var = a SLAM variable

value = a SLAM variable, or a mathematical expression

Commonly used SLAM variables:

ATRIB(I) attribute I of the current entity

XX(I) global variable I

TNOW the current time in the simulation.

The QUEUE block is provided by SLAM to give the packets a location in the network where they can wait to be serviced. The structure of the QUEUE block is as follows:

QUEUE(IFL),IQ,QC;

where,

IFL = file number (location of storage), an integer value

IQ = initial number of entities in queue

QC = maximum number of entities allowed in queue.

The QUEUE block must be followed by an ACT block, or ACTIVITY block. The ACT block represents servers or time delays in the system. The ACT block could be used to represent a communication link in so much as the link merely causes a time delay of the signal. The ACT block has the following structure.

ACT(N)/A,DUR,PROB or COND,NLBL;

where,

N = number of parallel servers

A = activity number

DUR = duration specified for the activity

PROB = probability specification for selecting the activity

COND = condition for selecting the activity

NLBL = end node label (required if end node is not the next node).

To destroy or delete the packets after they are finished in the system, or for whatever reason the user may have, the TERM or TERMINATE block is used;

TERMINATE,TC;

where,

TC = end simulation after TC entities arrive to the TERMINATE block.

The RESOURCE block declares resources. The AWAIT block, like the QUEUE block, is a place in the network where entities wait. However, the exit from an AWAIT block does not depend on a server becoming idle. Exit from the AWAIT block occurs when the resource specified is free. The FREE block releases a resource when an entity arrives to the block. A resource has a resource number associated with it. For example,

RESOURCE/THING1,1/THING2,2/THING5,5;

the resource number of thing1 is 1, of thing2 is 2, and the resource number of thing 5 is 3. SLAM automatically assigns resource numbers according to the order in which the resources are declared. The 1, 2, and 5 specify the file in which entities will wait for their resource. So, entities wait in file 1 for thing1 (resource 1) to be freed and they wait in file 5 for thing5 (resource 3) to be freed.

The user can collect statistics on five types of variables, outlined below.

COLCT,TYPE,ID,NCEL/HLOW/HWID,M;

TYPE = statistics can be collected on five types of variables:

= FIRST (coll. stat. on time of 1st arrival, one value is recorded during each run)

= ALL (coll. arrival time statistics for all entities)

= BETWEEN (time between the arrivals is collected as the statistic of interest)

= INT(NATR) (collect statistics relating to the arrival time of the entity minus the value of an atrib specified by NATR, the statistic equals TNOW-ATRIB(NATR))

= SLAM (the observation of a SLAM variable is recorded each time an entity enters the node)

ID = 16 character (or less) identifier associated with a particular COLCT node

M = number of branches

Histogram Parameters: (NCEL, HLOW, HWID)

NCEL = number of cells

HLOW = upper limit of the 1st cell

HWID = width of each cell e.g.,

(infinity,0],(0,10],(10,20],(20,+infinity)

NCEL = 4, HLOW = 0, HWID = 10 (4/0/10)

2.1.2 SLAM Statements

The statements required by SLAM are outlined below. These statements must be included in a SLAM model. The first statement in any SLAM simulation model is the GEN statement, this statement provides general information about a simulation. The structure of the GEN statement is given by,

GEN,NAME,PROJECT,MONTH/DAY/YEAR,NNRNS,ILIST,IECHO,IXQT,IPIRH,ISMRY/FSN,IO;

where,

NAME = 20 character field to identify the analyst

PROJECT = 20 character field to identify the project

MONTH/DAY/YEAR = date entered as integers

NNRNS = number of runs (default = 1)

ILIST = if YES a numbered listing of all input statements is printed including any error messages

IECHO = if YES an echo summary report is printed

IXQT = if YES execution is attempted if no input errors were

detected

IRIRH = if YES the heading INTERMEDIATE RESULTS is printed prior to execution of each simulation run

ISMRY/FSN = if ISMRY is YES the SLAM Summary Report is printed in accordance with the next field specification, FSN

IO = specifies the number of columns to be used for output reports.

The second statement in all SLAM models is the LIMITS statement. In the limits statement, the modeler specifys the number of SLAM files used, the maximum number of attributes used, and the maximum number of concurrent entries in the files. The SLAM files include those files used for QUEUE blocks and AWAIT blocks. The structure of the LIMITS statement is given by,

LIMITS,MFIL,MATR,MNTRY;

where,

MFIL = largest file number used

MATR = largest number of attributes per entity

MNTRY = maximum number of concurrent entries in all the files

MNTRY $\leq$ NNSET/(MATR+4)

NNSET = dimension of NSET/QSET (on the TISL VAX NNSET = 40000)

The NETWORK statement preceeds the network blocks in a SLAM model. So the modeler would place the NETWORK statement before any SLAM blocks (CREATE, ASSIGN, etc.). The network blocks are followed by the END statement. The structure of the NETWORK and END statements are as follows,

NETWORK,option,device;

if "option" = SAVE and "device" is set to a logical unit number: causes the decoded network to be written in binary form to the logical unit specified

```
END;
```

The INIT statement specifies the time the simulation is to begin and end. The INIT statement must be included in all SLAM models. The FIN statement signifies the end of the SLAM input statements (network blocks and statements), and is the last statement in all SLAM models. The structure of the INIT and FIN statements are given by,

```
INIT,TTBEG,TTFIN;
     TTBEG  =  beginning time
     TTFIN  =  ending time
FIN;
```

2.1.3 Example 2-1

In this example, a communication network is being simulated. This network has a single node and the channel is being modeled as a simple delay. The model could be expanded to include several nodes and a more complicated channel model. A listing of the SLAM code used to model this example is shown below in Figure 2-1.

In Example 2-1, packets are being generated at the CREATE block, after being created the packet will have its creation time placed in the first attribute. As a packet enters the ASSIGN block, its second attribute gets the value 1 placed in it and the third attribute gets the value 4096, and the global variable xx(1) gets set to 100. Then the packet enters the AWAIT block, the packet will remain there until a unit of the resource PACK1 is free. When the resource PACK1 is free, the previous packet has finished, and the current packet will leave the AWAIT block and immediately branch to the QUEUE node, labeled CHNL. The packet will have a zero wait time in the channel queue and will be serviced immediately in the ACT block. The ACT block

```
GEN,ED FRIEDMAN,EXAMPLE,3/14/85,1,NO,NO;
LIMITS,2,3,300;  two files used, three attributes per packet, three hundred
;                concurrent entries in the files
;
;       variables used:
;
;       atrib(1) = packet creation time
;       atrib(2) = node number
;       atrib(3) = packet length
;
;       xx(1) = mean service time in the channel
;
NETWORK;
;
      RESOURCE/PACK1,1;  resource 1 is PACK1
;-------------------------------------------------------------------------------
;
;       node model
;       ==========
;
      CREATE,EXPON(50,4),100,1;  create packets with an exponential interarrival
;                                time, with a mean of 50, generated using
;                                the random number stream 4, create the first
;                                packet at time 100, store the creation time
;                                in atrib(1)
      ASSIGN,ATRIB(2)=1.0,
             ATRIB(3)=4096.0,
             XX(1)=100.0,1;      assign attributes and the global variable
;
      AWAIT(1),PACK1;            packets wait at the node for the channel
;
      ACT,,,CHNL;                immediate branch to the channel
;
;-------------------------------------------------------------------------------
;
;       channel model
;       =============
CHNL  QUEUE(2);                  channel queue
;
      ACT/1,EXPON(XX(1),8);      exponential service time in the channel with a
;                                mean of xx(1) using the random number stream 8
;
      FREE,PACK1/1;              free one unit of resource 1 ( PACK1 ) when the
;                                packet is finished being serviced
;
      COLCT,INT(1),TIME DELAY;   collect system delay statistics, TNOW-ATRIB(1)
;
      TERM;                      terminate packets
;
      END;
;
INIT,0,1000;  run the simulation for 1000 time units
FIN;
```

Figure 2-1. SLAM code for Example 2-1, a simplified communications network.

represents the delay through the channel which is distributed exponentially with a mean of xx(1). When the packet leaves the ACT block, it enters the FREE block where one unit of the resource PACK1 is released and sent back to the AWAIT block which allows the next packet to begin the process. The output of Example 2-1 will be a SLAM Summary Report showing the system delay statistics, the file statistics, the service activity statistics, and the resource statistics. The file statistics include the AWAIT, QUEUE, and the event calendar statistics, see Figure 2-2.

The execution phase of a simulation begins by selecting the first event on the event calendar. The processor advances the current simulated time, TNOW, to the event time corresponding to this event. It processes the event by performing all appropriate actions based on the decision logic associated with the block type to which the entity is arriving. For example, if the entity is arriving to the AWAIT block, the decision logic involved with realizing the event consists of testing to determine if the resource is available; if yes, the entity seizes the resource and exits the node; otherwise, the entity is placed in the specified file and waits for the required resource. Although the decision logic is different for each node, the logic will result in one of three possible outcomes for the arriving entity:

1. The entity will be routed to another block;
2. The entity will be destroyed at the block; or
3. The entity will be delayed at the block based on the state of the system.

In this section, the Network Modeling approach of SLAM has been outlined so that the material in chapter 3 on the CSMA/CD simulation model can be understood without referencing [1]. The advantage in using the Network approach is that models can be developed quickly. With a basic understanding of

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT EXAMPLE BY ED FRIEDMAN

DATE 3/14/1985 RUN NUMBER 1 OF 1

CURRENT TIME 0.1000E+04
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
TIME DELAY	0.2490E+03	0.1286E+03	0.5166E+00	0.7331E+02	0.4032E+03	6

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	AWAIT	1.4883	1.3332	4	4	135.3010
2	QUEUE	0.0000	0.0000	0	0	0.0000
3	CALENDAR	1.8682	0.3382	3	2	47.9034

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
1	CHNL QUEUE	1	0.8682	0.3382	1	0.0000	100.0000	282.0977	6

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTILIZATION	CURRENT UTILIZATION
1	PACK1	1	0.8682	0.3382	1	1

Figure 2-2. The SLAM Summary Report, for Example 2-1, a simplified communications network.

the SLAM blocks available, a user can implement a simulation in very little time. Although useful, the Network approach has its limitations. For example, the user cannot remove entries from the event calendar, or remove a specific entry in a queue.

2.2 Discrete Event Modeling

Probably the greatest motivation for using the Discrete Event type of modeling is the ability to do file manipulations. The user can remove entries from a queue in any order desired, and can alter the event calendar. Suppose the modeler wants to remove an event which is due to occur at some later time so that the particular event will not occur, this can be done quite easily in a Discrete Event model, but is impossible to do in a Network simulation. For example, in the CSMA/CD model, packets enter the channel model and begin to transmit if the channel has been sensed idle. At this point an event is scheduled which signifies the end of the collision discrimination period (the time from the beginning of transmission to the point where all the other nodes realize that a node is transmitting). If a collision occurs, this event which signifies the end of the collision discrimination period must be removed from the event calendar.

Although this added flexibility is nice, it does not come without drawbacks. The Discrete Event models tend to be much more lengthy and somewhat more complicated than the Network models. If the user wanted to create packets without using the CREATE block, this would require several lines of code, whereas in the Network model it is simply one statement. This is the reason that combined Network/Discrete Event modeling is the preferred approach for most applications, and is the topic of section 2.3.

In this section, a brief overview of the Discrete Event modeling approach is given. In SLAM, many built-in subroutines and functions are provided, which relieve the user of much of the more complicated simulation aspects. For example, SLAM automatically causes the execution of code at the proper time, the user merely places an event on the event calendar, and SLAM will execute that event in the proper chronological order.

This section is arranged into six subsections. The first describes the INTLC, EVENT, and OTPUT subroutines, the second describes the SLAM subroutines, the third describes the SLAM functions, the fourth section describes the method of collecting statistics, and the fifth and sixth sections give examples.

2.2.1 The INTLC EVENT and OTPUT Subroutines

In Discrete Event models, the modeler creates a DRIVER file or SLAM INPUT FILE, and one or more FORTRAN files. The FORTRAN file(s) must contain an EVENT subroutine (required), an INTLC subroutine, and an OTPUT subroutine (INTLC and OTPUT are optional). The EVENT subroutine calls the user written events. The user written events may or may not use the subroutines provided by SLAM. The INTLC subroutine is used to initialize SLAM variables and user defined variables, and to schedule an event to occur at sometime after the beginning of the simulation. The OTPUT subroutine is used so that the modeler can have any kind of output desired, in addition to the usual SLAM Summary Report.

The DRIVER contains much of the same statements as a Network Model. That is, the GEN,LIMITS,INIT, and FIN statements must be in the DRIVER. For example,

```
GEN,ED FRIEDMAN,EXAMPLE,3/14/85,NO,NO;
```

```
        LIMITS,2,3,300;
        INIT,0,1000;
    FIN;
```

is a valid DRIVER file.

In the event subroutine, the events created by the modeler are called. If a specific model has three user written events called SENSE, TRANSMIT, and PROPAGATE, the event subroutine would look like the following,

```
        subroutine event(i)
        go to (1,2,3),i
1       call sense
        return
2       call transmit
        return
3       call propagate
        return
        end
```

The event code is the number to the left of the call statements. The event code is passed to the event subroutine when the proper time for execution is reached. Then the event subroutine calls the specific event as specified by the event code.

2.2.2 SLAM Subroutines

The SLAM processor relieves the user from chronologically ordering events on an event calendar. The user simply schedules an event to occur and SLAM causes each event to be processed at the proper time in the simulation. The events are scheduled using the schdl subroutine.

```
call schdl(kevnt,dtime,a)
```

kevnt = event code of the event being schedule, e.g., the event code of event sense above is 1, and that of propagate is 3

dtime = the number of time units from the current time, TNOW, that the event is to be processed

a = can be a user defined vector or the atrib array

The schdl subroutine is probably the single most important subroutine in the Discrete Event modeling approach. It allows the user to place events on the event calendar. If the user wants an entity to pass through a given block of code at a certain point in time, the schdl subroutine allows this to be done easily. Suppose, for example, that the user wants to model the beginning of transmission on an Ethernet type Local Area Network. After sensing the channel as idle, the packet must wait the interframe spacing before beginning transmission. This can be done by using the schdl subroutine as follows,

```
call schdl(2,9.6,atrib)
```

where,

2 = the event code of the transmit event

9.6 = the interframe spacing

atrib = the attribute array

In SLAM a file provides a way to store the attributes of an entity, the attributes will be stored according to some ranking with respect to the other entities in the file. Each entity in a file has a rank which specifies its position relative to the other members in the file. A rank of 1 denotes that the entity is the first entry in the file. The user can specify the ranking criterion, for example, first-in-first-out (FIFO) or last-in-first-out (LIFO) etc. If the ranking criterion is not specified, SLAM assumes FIFO.

The NNQ(IFIL) function returns the number of entries in a file. For example, x=nnq(1) sets x equal to the number of entries in file 1. To use a SLAM function, the user must declare the function as integer or real.

Like the QUEUE node, the filem subroutine stores entities in a file. For example,

```
atrib(1)=tnow
```

call filem(1,atrib)

Places an entry into file 1 with its first atrib set equal to the current time.

The rmove subroutine removes an entry with rank nrank from file ifile and places its attributes in the atrib array.

call rmove(nrank,ifile,atrib)

For example,

call rmove(2,3,atrib)

removes the second entry in file 3 and places its attributes in the attribute array

call rmove(nnq(1),1,atrib)

removes the last entry in file 1 and places its attributes in the attribute array

Note:

If the user attempts to remove an entry from a file with rank greater than nnq(ifile) a SLAM error will occur.

Consider the following example,

1. Creating packets from several nodes in a network.
2. Each packet has its second attribute set to the node number.
3. Suppose for some reason, the packets from all the nodes must be stored into a file using filem.
4. As the packets are removed from the file using rmove, the node that a particular packet originated from would be known by checking the value of the second attribute.

The copy subroutine causes the attributes of the nrank entry to be copied into the atrib array without removing the entry.

call copy(nrank,ifile,atrib)

For example,

```
call copy(10,5,atrib)
```

copies the attributes of the 10th entry in file 5 into the atrib array.

2.2.3 SLAM Functions

The mmfe(ifile) function sets a pointer to the first entry in file ifile. For example, x=mmfe(7), sets x to point at the first entry (rank 1) in file 7. The nsucr(ntry) function increments the value of the pointer ntry. For example, x=nsucr(x), will increment the pointer x, so that x now points to the next entry in a file. The nfind function can be used to determine the rank of an entry in a file which contains a value for a specified attribute that bears a relationship, designated by the user, to the value of a variable specified in the nfind function.

For example,

```
nrank = nfind(6,3,4,0,10.0,0.0)
```

is saying, search file 3 beginning with the 6th entry to determine the rank of the entry which has its 4th atrib exactly equal to 10, and place the rank in the variable nrank. The nfind function is explained in great detail in chapters 7 and 8 of [1].

2.2.4 Statistics Collection

To collect statistics based on an observation, the user must include a STAT statement in the DRIVER. The statistic will be collected by calling the SLAM subroutine COLCT.

```
STAT,ICLCT,ID,NCEL/HLOW/HWID;
```

ICLCT = is an integer number that the user specifies as

the file

for storing the statistics

ID = user specified identification

NCEL/HLOW/HWID = histogram parameters (see COLCT NODE in section 2.1.1)

```
Call colct(xval,iclct)
```

xval = the variable for which the statistics are being collected

iclct = as above

For example, the DRIVER file might look like,

```
GEN,ED FRIEDMAN,3/14/85,NO,NO;
LIMITS,2,3,300;
STAT,1,NODE 1 SYS TIME;
FIN;
```

2.2.5 Example 2-2 File Manipulations

Suppose for some reason, packets are being placed into file 1 using the filem subroutine.

```
call filem(1,atrib)
```

File the packet being processed into file 1.

And we want to remove a specific entry in file 1, using the nfind and the rmove subroutines.

```
rank = nfind(1,1,2,0,node,0.0)
call rmove(rank,1,atrib)
```

Set the variable rank equal to the rank of the first entry found in file 1 whose second attribute is exactly equal to the variable node. And remove that entry in file 1 and place its attributes in the attribute array.

2.2.6 Example 2-3 File Manipulations Involving the Event Calendar

The SLAM variable NCLNR is equal to the file number of the event calendar. Where, NCLNR=MFIL+1, recall MFIL from the LIMITS statement is the maximum number of files used. Like entites, events contain an atrib array, and the MATR+1, recall MATR from the LIMITS statement is the maximum number of attributes used, attribute of an event contains the event code (recall the event code of the sense event is 1 from above). The MATR+2 attribute of an event contains the event time, which was specified in the schdl subroutine. For example,

Suppose in the LIMITS statement, MATR is set to 3 and MFIL is set to 2. Then attribute 4 contains the event code, attribute 5 contains the event time, and the NCLNR variable is 3.

Suppose event 1 has been scheduled to occur 100 time units from the time when the schdl subroutine was called,

```
call schdl(1,100,atrib).
```

And suppose that this time in the simulation has not been reached. Further, suppose that for some reason, we do not want event 1 to be executed after all. For example, a packet in a CSMA/CD network has experienced a collision and we do not want the end of the transmission to be executed any more. So, event 1 must be removed from the event calendar. But first we must find event 1 in the event calendar using the mmfe, and nscur functions.

```
next = mmfe(nclnr)
```

set the variable next to point to the first entry in the event calendar

```
IO      call copy(-next,nclnr,atrib)
```

```
        copy the attributes of the entry pointed to by the pointer next
        into the attribute array, the negative sign indicates that next
        is a pointer
    if (condition)
        then this entry in the event calendar is the one we want to
        remove
    call rmove(-next,nclnr,atrib)
        remove this entry from the event calendar
        else the entry copied out of the event calendar was not the
        particular event that we want to remove
    next=nsucr(next)
        increment next to point to the following entry in the event
        calendar
    go to 10
        continue the process until we find the event to be removed
    end if
```

In this section, the Discrete Event modeling approach has been described. This approach allows the user to do many modeling techniques which were not allowed in the Network Modeling approach. These modeling techniques include file manipulations, an output report which is tailored to the modelers needs, and the manipulation of the event calendar. Although the Discrete Event approach is very useful to communication network modeling, it does become quite cumbersome for generating packets and simulating a simple queue. Therefore, it is very desireable to combine the Network and Discrete Event modeling approaches.

2.3 Combined Network/Discrete Event Modeling

By combining the two modeling approaches, Network and Discrete Event, the user can take advantage of both worlds. In combined modeling, it is advisable to make use of the Network approach as much as possible due to its simplicity and ease of model building. In the CSMA/CD simulation model, the Network approach was not used extensively due to its lack of flexibility.

In the CSMA/CD simulation model, two additional topics from SLAM were used that have not been covered yet. They are the EVENT block, which allows the modeler to link the Network model to the Discrete Event model, and the FREE subroutine. Like the FREE block in the Network approach, the FREE subroutine allows the modeler to free units of a resource. In this section, these last two topics will be discussed.

The structure of the EVENT block is shown below,

EVENT,JEVNT,M;

where,

JEVNT = specifies the event code of the event that the entity is being mapped into

M = the maximum number of emanating activities to be taken following the processing of the EVENT block.

By placing the EVENT block in the Network model, the modeler can map packets from the Network model into the Discrete Event model. The packets will be mapped into the event specified by the event code of JEVNT. At the current time in the simulation, TNOW, the event JEVNT will be executed for the packet which has just entered the EVENT block.

By using the FREE subroutine, the modeler can free units of resource from the Discrete Event model which will tell the AWAIT block in the Network model that the next entity or packet can begin the process. The structure of the FREE subroutine is as follows,

call free(ir,n)

where,

ir = resource number

n = number of units of resource ir to be released

In this chapter, a brief introduction to the simulation language SLAM has been given. This introduction is intended as a primer for Chapter 3 where the CSMA/CD simulation model, implemented at the Telecommunications and Information Sciences Laboratory, is explained fully. The full description of SLAM can be found in [1], in addition a set of notes which were prepared for a seminar are also available to the interested reader [2].

3.0 THE CSMA/CD SIMULATION MODEL

The Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol that was proposed by Metcalf and Boggs [3], has become known as Ethernet. The Ethernet has since evolved [4] and is described in the IEEE Standard 802.3 [5]. The Ethernet was designed and implemented as a joint effort by Digital Equipment Corporation, Intel Corporation, and the Xerox Corporation. The Ethernet is a broadcast network with a long coaxial cable. The cable is typically distributed through a building, and the terminals tap onto the cable. In this study, the Ethernet standard need not be followed, the main concern is to accurately simulate a general CSMA/CD protocol.

This chapter is divided into three sections. The first section gives an introduction to the CSMA/CD protocol. The second section, which is the main topic of this chapter, gives a complete description of the CSMA/CD simulation model implemented at the University of Kansas Telecommunications and Information Sciences Laboratory (TISL). The third section compares the simulation model to the published performance results of the Ethernet.

The comparison will be made with the published results of Shoch [6, 7, 8]. The results published by Shoch were taken from an acutal Ethernet Local Area Computer Network. In addition, a comparison will be made to the results published by Hughes and Li [9]. The study done by Hughes and Li was a simulation also, and allows a comparison of system delay. There was also a study done by Acampora et.al. [10], which gives another comparison of delay. By comparing the simulation results of the model implemented to the published results, it is clear that the simulation accurately models CSMA/CD networks.

The main goal of this chapter is to introduce the reader to the CSMA/CD Local Computer Network simulation model. Together with the introduction to the simulation language SLAM, presented in Chapter 2, the reader should be

well equipped to understand and/or modify the CSMA/CD simulation model. The tool used to model the CSMA/CD protocol was described in Chapter 2, namely SLAM, and in this chapter, the actual model will be described in detail.

3.1 Review of the CSMA/CD Protocol

The CSMA/CD protocol can be described as listen-before and listen-while transmitting. That is, before beginning transmission a network node will listen to the channel to see if any other nodes are transmitting. If another node is transmitting, then the node will defer or wait until the channel is idle. When the channel is sensed idle, the node will begin to transmit after waiting the interframe spacing, which is typically 9.6 microseconds [5]. While transmitting, the node will monitor the channel to make sure the packet or message is received without error.

When two or more nodes attempt to transmit their packets at the same time errors occur. When this happens, it is said that a collision has occurred. Clearly, there is a finite amount of time in which collisions can occur. This amount of time is known as the collision discrimination period. The collision discrimination period is the time period from the beginning of transmission to the point in time where all other nodes will see the transmission and will defer their transmission. The collision discrimination period is set in the model and is one round-trip propagation, typically 5 to 10 microseconds. The packet length should be greater than the time required to travel one round-trip propagation. That is, the packet length should be greater than the collision discrimination period.

If a collision does occur, the node will send a jam signal and calculate a backoff time. This jam signal is used to inform all other transmitting nodes that a collision has occurred and they should stop transmitting. When a

jam signal is seen, the transmitting nodes will determine a backoff. The backoff is a randomly chosen amount of time which is used to determine the amount of time to be waited before attempting to transmit again. When a collision is detected the node sends a jam signal, calculates a random wait time or backoff, waits that backoff time, and begins the process again.

The backoff can be determined in any manner seen fit by the analyst. In the CSMA/CD model described here, and in the Ethernet, a truncated binary exponential backoff scheme is used. In this scheme, the backoff is determined by first sampling a uniform distribution, and then taking that random sample and multipling it by the slot time. The slot time is approximately the round-trip propagation delay. The size of the uniform distribution is dynamic and depends on the number of retransmission attempts. The lower bound of the uniform distribution is zero, and the upper bound is equal to

$$2^n-1$$

where n is the number of retransmission attempts. If n becomes larger than some value, typically 10 [5], then the upper bound is truncated to

$$2^{10}-1.$$

The decision logic for the CSMA/CD protocol can be seen by steping through the flow diagram shown in Figure 3-1.

3.2 The TISL CSMA/CD Simulation Model

In this section the CSMA/CD simulation model developed at the University of Kansas Telecommunications and Information Sciences Laboratory (TISL) will be explained. The model was thoroughly validated and accurately simulates a real system, section 3.3 will discuss the validation results. The model is divided into two distinct parts. The first part is the Network Model, where SLAM Network blocks are used to simulate the arrival of packets to the network

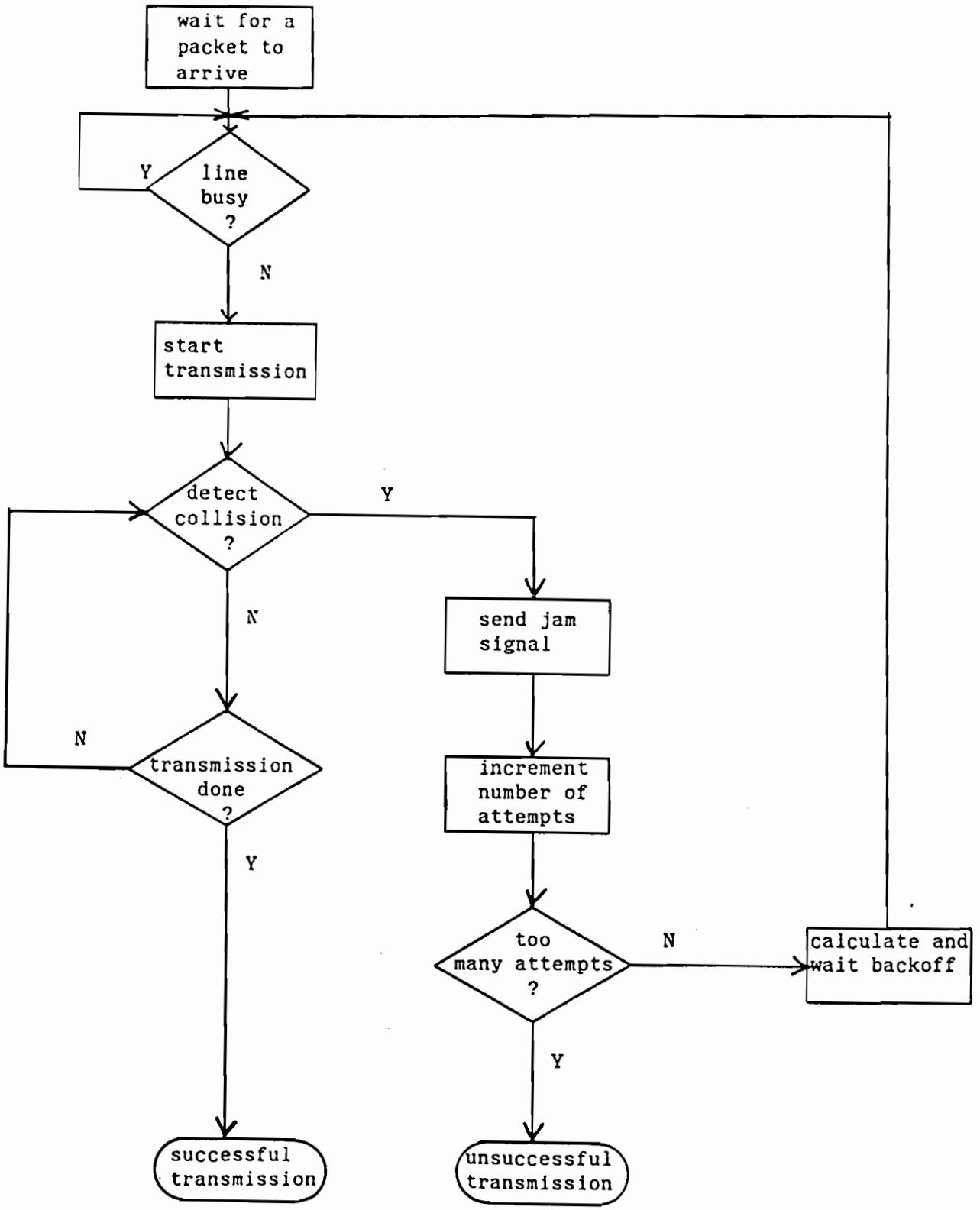


Figure 3-1. The decision logic for the CSMA/CD protocol.

and the build up of packets at the node queues. The second part is the Discrete Event Model, where the CSMA/CD protocol and the propagation on the line are simulated.

Section 3.2 is divided into six subsections. The first will describe the variables and files used. The second will describe the Network Model. The third will describe the Discrete Event Model. The fourth describes how the simulation is initialized. The fifth describes the output. And the sixth describes how the system configuration can be modified. All of the SLAM variables, subroutines, and blocks used to model the CSMA/CD Local Area Network were described in Chapter 2.

3.2.1 Variables and File Assignments

The variables used to model the CSMA/CD Network are divided into four types. The first variable type used are the SLAM variables. The second are model variables, the third are CSMA/CD variables, and the fourth are measurement variables.

3.2.1.1 SLAM Variables

The SLAM variables include the attributes, TNOW, NCLNR, and global variables. The TNOW variable is the current time in the simulation. The NCLNR variable is the file number of the event calendar. The attributes are an array of variables carried with the packet as it passes through each process in the model. The attribute array of each packet is really a list of variables, each packet has the same definition of a particular attribute, while the actual value for the attribute may be different. The definition of each attribute is listed below.

atrib(1) = packet creation time

atrib(2) = node identification

atrib(3) = propagation left marker, if left prop is finished then atrib(3) = 0

atrib(4) = propagation right marker, if right prop is finished then atrib(4) = 0

atrib(5) = if atrib(5) = 0 then the event being scheduled is propagation, if atrib(5) = 1 then the event being scheduled is transmission

atrib(6) = counter for the number of attempts to transmit

atrib(7) = end of propagation left marker, if atrib(7) = 0 then finished prop left

atrib(8) = end of propagation right marker, if atrib(8) = 0 then finished prop right

atrib(9) = packet length in microseconds

atrib(10) = atrib(9)-slttim = amount of time left after the collision discrimination period

atrib(11) = number of bits per packet

atrib(12) = marks the time transmission is attempted, gets reset each time a retransmission is attempted

atrib(13) = while packet is in the AWAIT block atrib(13) = 0, set to 1 when packet leaves the queue, used to collect statistics on the amount of time a packet is queued, tqueud=atrib(14)-atrib(1) if atrib(13) = 0

atrib(14) = marks the time when a packet leaves the node queue (the AWAIT), used to collect statistics on the time required to access the network, taccss=atrib(12)-atrib(14)

atrib(15) = not used

atrib(16) = not used

atrib(17) = first access attribute, if the packet is trying to transmit for the first time then atrib(17) is set to 1 from 0, on subsequent attempts to transmit atrib(17) is incremented

3.2.1.2 Model Variables

The model variables are those which were created to simulate certain aspects of the CSMA/CD protocol. The three most important model variables are the node state array, the station delay array, and the network status array. The node state array is dimensioned to the maximum number of stations (maxsta). Each element in the node state array describes the state that a particular node is in. For example, if element 5 of the node state array contains the number 5, then the fifth node on the network is idle. There are seven possible node states and they include all the possible states that a node can be in. They are, defer, transmitting, collision discrimination period, jamming, terminating, idle, and collision during interframe spacing. The node state array and the possible states are listed below.

node states:

idefer = 0 = node is defering transmission

itrans = 1 = node is transmitting

icolpr = 2 = node is in the collision discrimination period

ijamng = 3 = node is jamming

iterm = 4 = node is terminating a packet

iidle = 5 = node is idle

intrmv = 6 = node has experienced a collision during the interframe spacing

nodest(maxsta) = node state array, each element is set to one of the

states above, and defines the state of a particular node

maxsta = maximum number of nodes (or stations) being simulated

The station delay array contains the spacing between nodes in time. The station delay array is dimensioned to the maximum number of nodes minus one (maxsta-1). Each element of the station delay array contains a spacing. For example, element one contains the spacing between nodes 1 and 2, element two contains the spacing between nodes 2 and 3, and element (maxsta-1) contains the spacing between nodes (maxsta-1) and maxsta.

stadly(maxsta-1) = distance between nodes (in microseconds), e.g., stadly(1)=distance between nodes 1 and 2

The network status array gives an indication of the way a particular node sees the channel. Like the node state array, each element in the network status array corresponds to a particular node. The array is dimensioned to the maximum number of stations. The network status array is used to sense the channel. If an element in the array contains a 0, then the node corresponding to that element sees the channel as idle. If an element in the array contains a 1, then the node corresponding to that element sees the channel as busy. If an element in the array contains a 2 or greater than the node corresponding to that element sees the channel as undergoing collisions. As will be discussed later, when the packet is traveling down the line the element in the network status array that corresponds to the node that the packet has entered will be incremented. The network status array is outlined below.

ntstus(maxsta) = network status array, if 0 then channel is idle, if 1 then channel is busy, if greater than 1, then channel is undergoing collisions

3.2.1.3 CSMA/CD Variables

The third variable type are the CSMA/CD variables. These are variables particular to the CSMA/CD protocol. These variables include the maximum allowable number of collisions (mxcoll). If greater than mxcoll collisions occur, the packet will be discarded by the simulation. In a real network, the message being discarded will have to be retyped and sent again. The wait time or interframe spacing (waitim) is an amount of time that a packet will wait before beginning to transmit. When the channel is sensed idle, the protocol says to wait a small amount of time, called the interframe spacing, before beginning to transmit. The jam time is another CSMA/CD variable, it is the amount of time required to transmit the jam signal. Recall that the jam signal is sent when a collision occurs to inform all the transmitting nodes that a collision has occurred, and the nodes must calculate and wait a backoff time. The jam signal represents wasted time and should be made as small as possible. The slot time is the fourth and final CSMA/CD variable. The slot time is the collision discrimination period. That is, the slot time equals the amount of time required for all other nodes to see that some other node is transmitting. The CSMA/CD variables are outlined below.

mxcoll = the maximum number of collisions = 16

waitim = interframe spacing = 9.6 microseconds

rjmtim = jam time

slttim = slot time = round trip propagation delay

3.2.1.4 Measurement Variables

The fourth and final variable type are the measurement variables. These variables are used to measure statistical information on the network. Many of the measurement variables are used to measure performance of the network, such

as the throughput. In addition to these measurement variables, many of the attributes are used to measure the performance of the network. The measurement variables are self explanatory and have been outlined below.

icnt(maxsta)	=	number of packets successfully and unsuccessfully transmitted from each node
icoll(maxsta)	=	counts the collisions experienced by each node
excoll(maxsta)	=	number of packets from each node that were discarded due to excessive collisions
frstat(maxsta)	=	number of packets successful on the first attempt to access the network
timegd(maxsta)	=	time spent sending packets successfully from each station, timegd/(total time) = throughput
attempts(mxcoll)	=	counter array, used to keep track of the number of packets successfully transmitted after 1 attempt, after 2 attempts, ..., and after mxcoll attempts
bitsgd	=	number of bits successfully transmitted
bitsbd	=	number of unsuccessful bits

3.2.1.5 File Assignments

There are two types of files in the CSMA/CD simulation model. The first type are the node queues. These are files which are used to store packets in the station queue. The node queues are really AWAIT blocks and will be described in the next section. Each node on the network has a station queue or AWAIT file associated with it, where packets will wait until the previous packet has finished transmission. The second file type are the statistics collection files. These files store the statistical information being collected. The analyst can specify anything of interest to be collected as a sta-

tistic, and SLAM will print out the information in the SLAM Summary Report. The file assignments have been listed below.

Files 1-15	Reserved for the AWAIT blocks at each station
Files 16-21	Not used
File 22	Channel QUEUE block
File 23	Not Used
File 24	Defer file
File 25	Event Calendar

In addition to the file types mentioned above, there are two other files that have been used. They are the event calendar and the defer file. The event calendar has been discussed in Chapter 2 and is numbered NCLNR. The defer file stores packets that are deferring from a busy channel. When the channel is sensed busy or undergoing collisions, the packet will be stored in the defer file, the defer file is numbered (NCLNR-1).

3.2.2 The CSMA/CD Simulation Model - Network Model

The CSMA/CD simulation model is divided into two parts. The first part is the Network Model and the second part is the Discrete Event Model. The Discrete Event Model will be discussed in section 3.2.3, and the Network Model will be discussed here. A reduced version of the Network Model is shown in Figure 3-2, and the complete listing is in Appendix A.1. The Network Model is explained in this section by stepping through the code listed in Figure 3-2.

In the Network Model, the arrival of packets to the network and the build-up of packets at the network stations is simulated. The simulation of packets arriving to the network is done using the SLAM CREATE block, lines 20, 27, and 37. Recall from section 2.1 that the CREATE block generates entities at a user specified rate. The entities in the case of the CSMA/CD model are

```
1   GEN,TISL,CSMA CD,8/20/84,1,NO,NO;
2   LIMITS,24,17,200;
3   STAT,1,NODE 1 SYS.  TIME
4   STAT,2,NODE 2 SYS.  TIME
5             .
6             .
7   STAT,15,NODE 15 SYS TIME
8   STAT,16,QUEUEING DELAY
9   STAT,17,ACCESS DELAY
10  STAT,18,CHANNEL DELAY
11  STAT,19,TOTAL SYS DELAY
12  NETWORK;
13        RESOURCE/PACK1,1/PACK2,2/PACK3,3/PACK4,4/PACK5,5/PACK6,6/
14                 PACK7,7/PACK8,8/PACK9,9/PACK10,10/PACK11,11/
15                 PACK12,12/PACK13,13/PACK14,14/PACK15,15;
16  ;
17  ;     MODEL OF THE NUMBER OF STATIONS ON THE LINE
18  ;     ==========================================
19  ;-----------------------------------------------------------------------------
20        CREATE,EXPON(13931.97265625),,1;create packets at node 1
21        ASSIGN,ATRIB(2)=1.0,
22               ATRIB(9)=1393.197265625,
23               ATRIB(11)=4096.0,1;        assign attributes
24        AWAIT(1),PACK1;                   packets wait for previous packet to
25        ACT,,,CHNL;                       finish immediate branch to the channel
26  ;-----------------------------------------------------------------------------
27        CREATE,EXPON(13931.97265625),1500,1;
28        ASSIGN,ATRIB(2)=2.0,
29               ATRIB(9)=1393.197265625,
30               ATRIB(11)=4096.0,1;
31        AWAIT(2),PACK2;
32        ACT,,,CHNL;
33  ;-----------------------------------------------------------------------------
34                  .
35                  .
36  ;-----------------------------------------------------------------------------
37        CREATE,EXPON(13931.97265625),29000,1;
38        ASSIGN,ATRIB(2)=15.0,
39               ATRIB(9)=1393.197265625,
40               ATRIB(11)=4096.0,1;
41        AWAIT(15),PACK15;
42        ACT,,,CHNL;
43  ;-----------------------------------------------------------------------------
44  ;
45  ;     COMBINE STATIONS TO FORM CHANNEL
46  ;     ================================
47  CHNL  QUEUE(22);                  DUMP ENTITIES INTO QUEUE
48        ACT;                        IMMEDIATE BRANCH TO CHANNEL MODEL
49        EVENT,1;                    GATEWAY TO DISCRETE EVENT MODEL
50        ASSIGN,XX(25)=10000000;     SET XX(25) TO TTFIN
51        END;
52  INIT,0,10000000;
53  FIN;
```

Figure 3-2. Shortened Version of Network Model.

packets. The rate at which packets are created is specified as a percentage of the load on the network. The nodes are generating packets with a Poisson arrival rate, which implies an exponential interarrival time. That is, if we assume that the system has a large number of independent packets which arrive to the system and pass through the process, then we can assume an exponential interarrival probability [11]. An exponential interarrival means that the time between the arrival of packets is distributed exponentially, and has a probability density described by equation (3-1).

$$a(t) = h\, e^{-ht} \qquad t > 0 \qquad (3\text{-}1)$$
$$= 0 \qquad t < 0$$

where,

a(t) = interarrival time probability density function

1/h = mean interarrival time

t = interarrival time

If the interarrival time is exponential, then the arrivals are Poisson. That is, the probability of exactly n packets arriving during an interval of length t is given by the Poisson law, equation (3-2) [11].

$$P_n(t) = [(ht)^n /n!]e^{-ht} \qquad (3\text{-}2)$$

where,

h = mean arrival rate.

The mean interarrival time is calculated as a function of the percentage of the load, the packet length, the total line capacity, and the number of nodes. In equation (3-3), the mean interarrival is given as a function of these parameters.

$$u = (NP)/(LC) \quad \text{seconds per packet} \qquad (3\text{-}3)$$

where,

u = the mean interarrival time

N = number of nodes whose combined load on the network simulates L percentage of the total capacity C

P = the packet length in bits per packet

L = offered load, percentage of the total capacity that is to be simulated in decimal

C = the total capacity of the line in bits per second.

The discussion above shows that the simulation of packets arriving to the network nodes is done using the SLAM CREATE block, and the formula used to calculate the mean packet interarrival time is given. After being generated, the packets pass through an ASSIGN block, lines 21-23, 28-30, and 38-40. In the ASSIGN block, the second, ninth, and eleventh attributes are set. Recall from above that the second attribute, atrib(2), gives the node identification. So, the first node on the network will have atrib(2) set equal to 1, the second node will have atrib(2) set equal to 2, and the maxsta node on the network will have atrib(2) set equal to maxsta.

The ninth attribute, atrib(9), sets the packet length in microseconds. To convert the packet length from bits per packet to seconds per packet, divide the packet length in bits by the total line capacity. Atrib(9) can be set to the same value for each node if the system being modeled has a constant packet length, or to different values if individual nodes generate packets with different lengths. The analyst can study a system in which packet lengths vary randomly by setting atrib(9) to a random distribution. The eleventh attribute, atrib(11), is the packet length in bits and the same comments made concerning atrib(9) apply to atrib(11).

After the packet is generated using the CREATE block and gets its attributes set in the ASSIGN block, the packet enters an AWAIT block, lines 24, 31, and 41. The AWAIT block represents a station queue. Packets will wait in

this block until the previous packet has finished its transmission or is discarded. Recall from section 2.1 that the entities or packets will wait in an AWAIT block until a unit of resource is freed. Every node being simulated must have a resource associated with it. When the packet leaves the AWAIT node, it carries its resource and will release the resource when terminated. The resources are defined using the RESOURCE statement on lines 13-15, the resources are named according to the node they belong to. For example, the resource associated with node 1 is PACK1, the resource associated with node 2 is PACK2, and so on.

When the previous packet finishes its transmission, the resource is released and the next packet leaves the AWAIT block, and enters an ACTIVITY block, line 25, 32, and 42. If there are no packets in the AWAIT block, the resource will remain at the AWAIT block until a packet arrives. When a packet arrives, it will seize the resource and proceed to the ACTIVITY block. In the ACTIVITY block, the packet is branched out of the Network Model and immediately enters the Discrete Event Model or Channel Model. Before branching into the Discrete Event Model, the packets are sent to the channel queue, labeled CHNL, line 47. The channel queue is used to avoid sending two packets into the EVENT block at the same time. This queue will not have any packets waiting in it since the following block is an ACTIVITY which has a zero service time, line 48. After exiting the ACTIVITY block at line 48, the packets enter the EVENT block, line 49, which branches them into the first event in the Discrete Event Model. The first event is SENSE, so the packet will immediately sense the channel upon leaving the Network Model. The Discrete Event Model will be discussed in section 3.2.3.

The discussion above shows how packets are generated and follows the flow through the Network Model. The other SLAM statements and blocks were discus-

sed in Chapter 2 and will briefly be discussed here. The first two statements are the GEN and LIMITS statements, lines 1 and 2. Recall that the GEN statement gives general information about the simulation such as the analyst's name and the date. The LIMITS statement is used to specify the number of files, attributes, and the maximum number of concurrent entries in all the files. These values are specified as 24, 17, and 200 respectively.

Following the LIMITS statement is a group of STAT statements, lines 3-11, which are used to specify the statistical information that is to be printed in the SLAM Summary Report. The first 15 STAT statements (3 through 14 are not shown) define the STAT files 1-15 to be used for collecting statistics on the total system delay for each node, individually. The next four STAT statements define the STAT files 16-19 which are also delay. However, they are the delays combined for all the nodes. They are the queueing delay, the access delay, and channel delay, and the total system delay. The queueing delay is the time that packets spend in their node queues, the AWAIT block. The access delay is the time spent trying to send the packet successfully, it begins when the packet leaves the node queue and ends when the node begins to send the packet and the packet is successful. The channel delay is the time required to send the packet over the channel, and is equal to the packet length in seconds. The total system delay is the time a packet is in the system. It begins when the packet is created and ends when the packet has either finished transmission or is declared unsuccessful. The total system delay is also the combination of the other three.

TOTAL SYS = QUEUEING + ACCESS + CHANNEL

The next statement, after the STAT statements, is the NETWORK block, line 12. The NETWORK block tells SLAM that all the code that follows will be SLAM network blocks. The RESOURCE block is the next statement, lines 13-15, and

was discussed above. The other code has already been explained, the groups of code, lines 20-25, and lines 27-32, and lines 37-42, simulate the nodes on the line. Again, lines 47-49, accomplish the branching of packets into the Discrete Event Model. On line 50, there is an ASSIGN block, this is used to set the global variable XX(25) to the end of simulation time (TTFIN). The following block is the END block, which tells SLAM that there will not be any more Network blocks.

The last two lines are the INIT and FIN statements, respectively. The INIT statement specifies the amount of time that is to be simulated. In this example, the simulation is running from time zero to ten seconds. The units of time are microseconds, so the value 10,000,000 for the end of the simulation is only ten seconds of real time. The FIN statement is the last line of code, the FIN statement denotes the end of all SLAM input statements.

In this section, it was shown how the mean value for the interarrival time of packets is calculated. The SLAM code used to simulate the arrival of packets to the network and the build-up of packets at the nodes, has been discussed. The simulation of network nodes was described. And the flow of packets through the Network Model and into the EVENT block which branches the packets into the Discrete Event Model has been presented. In the Network Model, the simulation of packets arriving to the network and the build-up of packets at the nodes can be done. The rest of the CSMA/CD protocol discussed in section 3.1 must be simulated using the Discrete Event Modeling techniques discribed in Chapter 2. In the following section, the Discrete Event Model used to simulate the CSMA/DC protocol, is discussed.

3.2.3 The CSMA/CD Simulation Model - Discrete Event Model

The Discrete Event portion of the CSMA/CD simulation model will be discussed in this section. Part of this model has already been discussed when the variables were explained. The node state, station delay, and network status arrays make up an important concept in the Discrete Event Model. The other major portion of the model is simulating the propagation down the line.

In the Network Model, we saw 1) how the simulation of packets arriving to the network is done; 2) that the packets will wait at the nodes until the previous packet has finished transmission; and 3) how the packets are branched into the Discrete Event Model. To simulate the rest of the CSMA/CD protocol, outlined in Figure 3-1, the Discrete Event approach to simulation that was described in Chapter 2 must be used. The network elements that are left to simulate are sensing the channel, checking for collisions, handling the backoff, and propagation on the line. The propagation presents some of the more challenging problems. If the node that originated the packet is somewhere in the middle of the line, the packet must propagate in both directions until it encounters the end of the line. If the packet originates at one of the ends of the line, propagation is in just one direction.

The node state array has an element for each of the nodes on the network. For example, element 1 is a storage location which holds the state that node 1 is currently in, and element 2 holds the state that node 2 is currently in, and so on. The node state array is used to describe where a particular node is in the protocol of Figure 3-1. There are seven possible states that a node can be in. The node can be defering transmission. If the node senses the channel as busy, the packet will be placed into the defer file which is numbered (NCLNR-1), and the node will be in the defer state. The node could be in the transmit state, which would indicate that the node has sensed the

channel idle and has begun to transmit the packet. The node may be in the collision discrimination state, which would mean that the packet is being transmitted but all the nodes on the line have not seen the leading edge of the packet arrive to their location so collisions could still occur. If the node is in the jamming state then that particular node has had a collision occur and is sending a jam signal. When a node is in the terminate state, its packet has finished transmission, and if a packet is waiting in the node queue it can now be released. When a node does not have any packets in the network, it is placed in the idle state. If a node senses the channel idle, it will begin to transmit its packet after waiting the interframe spacing, but the leading edge of a packet from another node may arrive to the node during this time. If this occurs, a collision will be declared and the node will be placed in a state to indicate this.

The network status array also has an element for each node in the network. The elements give an indication of how a particular node sees the channel. The channel may be idle, busy, or undergoing collisions. In the network status array element 1 contains node 1's view of the channel, and element 2 contains node 2's view of the channel, and so on. If an element in the network status array contains a 0 then the particular node that sees the channel as represented by that element will see the channel as idle. Similarly, if the element contains a 1 then the node sees the channel as busy, and if the element contains a 2 or greater then the node sees the channel as undergoing collisions.

The third array is the station delay array. It contains the spacing between nodes. The spacing has been converted into units of time. The time units in the CSMS/CD simulation model are microseconds. So, if a 635 meter line is being simulated, this distance must be converted into microseconds.

This requires that the velocity of propagation in the coaxial cable be known. The velocity will depend on the cable being used. For the Ethernet a minimum velocity of propagation is specified as 0.77c [5]. Which is approximately 2.31×10^8. A 635 meter line would be converted to a 2.75 microsecond line. The nodes will be spaced accordingly on the 2.75 microsecond line. The nodes can be equally spaced or irregularly spaced depending on the particular network to be studied. Each element of the station delay array contains the spacing between the nodes. For example, element 1 contains the spacing between nodes 1 and 2, and element 2 contains the spacing between nodes 2 and 3, and so on.

These three arrays make up the essense of the Discrete Event portion of the CSMA/CD model. The model consists of eight events. The events are either performing a certain portion of the protocol or handling the propagation on the line. In addition to the eight events, there are five subroutines. These subroutines are used to handle the collisions, remove packets from the defer file, and free a unit of resource when the packet has either finished its transmission or is declared unsuccessful. The event codes, event names, what events are scheduled from a particular event, and what subroutines are called by a particular event are listed in Table 3-1. The subroutine names, what events are scheduled from the subroutine, and what subroutines are called from the subroutine are listed in Table 3-2.

In this section, each of the events and subroutines will be described. The actual code developed will not be stepped through due to its length. The code will be described using flow diagrams of each event and subroutine. The actual code is listed in Appendix A.2. The code itself is fully commented and the interested reader should not have any trouble understanding it, so long as the reader has a basic knowledge of the computer programming language FORTRAN

77. The events and subroutines will be described in the order they appear in Tables 3-1 and 3-2. In the code, the events are written as FORTRAN subroutines. However, they represent events to SLAM.

Event Code	Identification	Schedules To	Subroutines Called
1	sense	transmit	
2	transmit	leftprop rigtprop success	collision chnlecho
3	leftprop	leftprop	collision chnlecho
4	rigtprop	rigtprop	collision chnlecho
5	success	endtrans	
6	endtrans	ltfinprop rtfinprop	freersc chnlecho
7	ltfinprop	ltfinprop	exitdefer chnlecho
8	rtfinprop	rtfinprop	exitdefer chnlecho

Table 3-1. The events used to simulate the CSMA/CE Local Computer Network

User Written Subroutine	Schedule To	Calls To
collision	endtrans	search calc wait backoff
search		
calc wait backoff	sense	
exitdefer	sense	
freercs		
chnlecho		

Table 3-2. The subroutines used to simulate the CSMA/CD Local Computer Network

The first subroutine to describe is the event subroutine. Recall that all Discrete Event models must have an event subroutine. An event will be scheduled to occur at some time in the future using the schdl subroutine provided by SLAM. Actually, the schdl subroutine places events on the event calendar, and at the proper time in the simulation, the event is pulled off the calendar, and the event subroutine does a FORTRAN call to the particular event which has been scheduled to occur. The event subroutine is merely used to call the proper event when the correct time in the simulation has been reached. The event subroutine does not model any portion of the protocol, it is merely a formality which must be taken care of in the simulation.

In the flow diagrams, there are some symbols and general techniques that will be described here. When an attribute is being changed or modified, the letter "a" will be used as an abbreviation of atrib. For example, attribute 17 would appear as atrib(17) in the code of Appendix A.2, and would appear as a(17) in the flow diagrams. As with standard flow diagrams, the diamond indicates a decision. The boxes may be comments or perform a specific function. Since the emphasis here is on the simulation of the network, all of the statistics collection operations have been omitted from the flow diagrams. In all of the events, the first thing that is done is to set the identification or ID, inode is set equal to the second attribute. This is done so that the node can be referenced using a variable that describes the node instead of using atrib(2).

3.2.3.1 Description of Events

In this section, the events created to model the CSMA/CD network will be described. Each of the eight events listed in Table 3-1 will be described

using flow diagrams. The first event that a packet must execute when it leaves the Network Model is the SENSE event (event code = 1).

In the SENSE event, like all the other events and subroutines, the first thing done is setting inode equal to the ID, atrib(2). Then the 10th attribute is set to the amount of time needed to transmit the packet after the collision discrimination period has expired. Once the collision discrimination period is over the packet being transmitted controls the channel and can finish the transmission successfully. The next operation performed in SENSE is to see if the channel is idle or busy. This is done by observing the value of the network status array element used by the ID node, ntstus(id). If ntstus(id) is zero, then the channel is idle, if not, then the channel is busy.

As seen in the flow diagram of Figure 3-3, there is a separate set of operations that must be performed depending on whether or not the channel is idle. If the channel is idle, the packet will begin transmission. If the channel is busy, the packet must be stored in the defer file. Whether or not the channel is idle, the fifth attribute is set to 1. This is done to indicate a transmission event. It will be seen later that it must be known whether an event scheduled is a propagation or a transmission. If the event is propagation, then atrib(5) is set to 0, otherwise it is set to 1. The reason this must be done is that for a certain set of circumstances an event will be removed from the event calendar but propagation events will never be removed. So to be sure that a propagation event is not removed the fifth attribute is used to signify propagation.

If the channel is idle, ntstus(id) = 0, then the node state must be set to transmitting, and the TRANSMIT event is scheduled to be executed after waiting the interframe spacing. If the channel is not idle, then the node

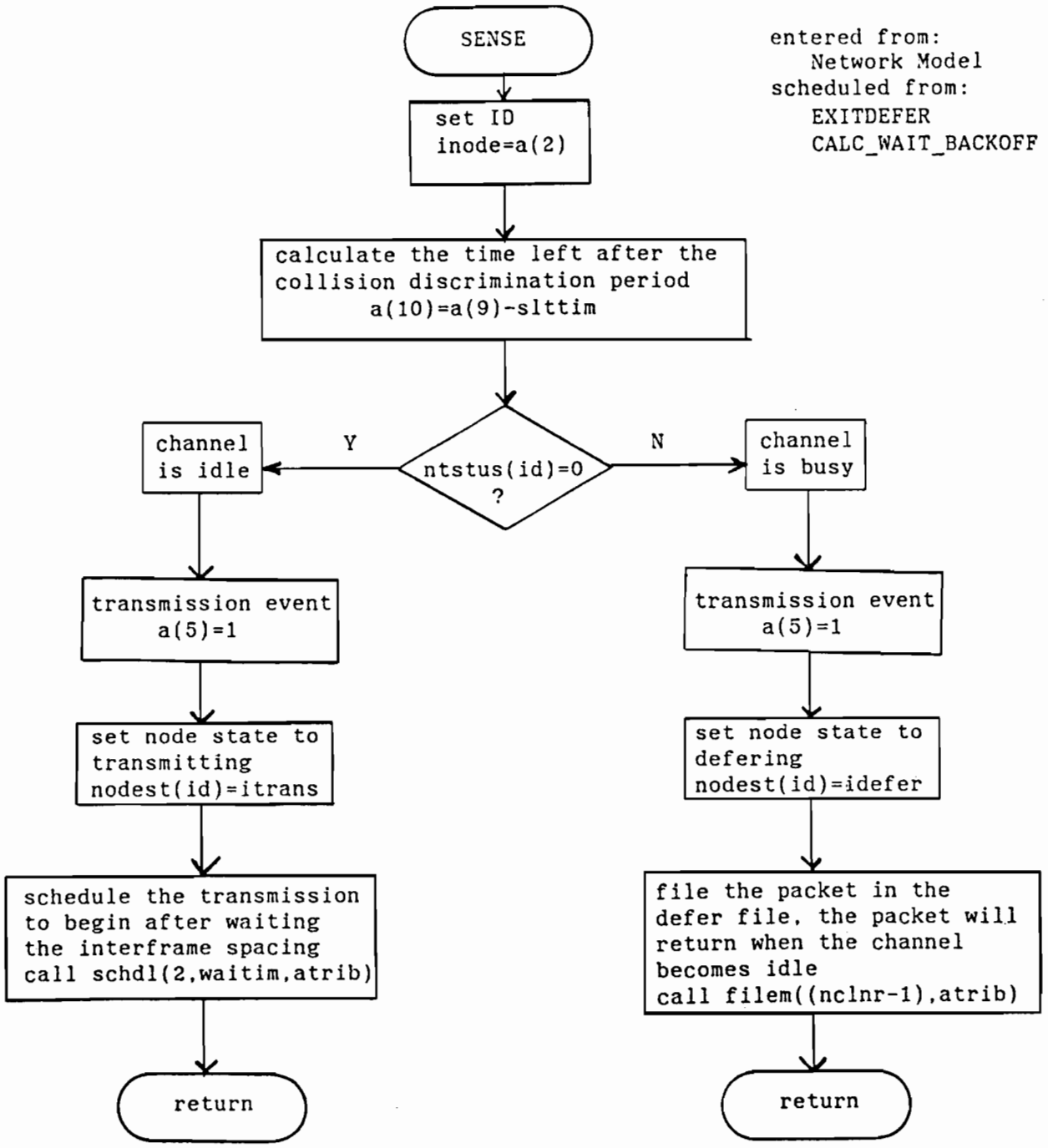


Figure 3-3. Flow diagram of the SENSE event.

must be placed in the defer state, and the packet is filed into the defer file using the SLAM subroutine filem. The defer file is numbered (NCLNR-1), where NCLNR is the file number of the event calendar.

If the channel is sensed idle, then the packet waits the interframe spacing and begins to transmit the packet by scheduling the TRANSMIT event (event code = 2). The first thing done in TRANSMIT is to increment the value of the network status array at the ID node. This changes the ID node's view of the channel from idle to busy. In the TRANSMIT event, the propagation to the nearest node to the left and right are done if the node is not at either end of the line. This is done by checking the ID, if the ID is 1 or maxsta then the node is at the beginning or end of the line, respectively, and if the ID is not 1 or maxsta then the node is somewhere between the two ends and the packet must be propagated in both directions. If the ID node is at the beginning of the line the propagate left marker, atrib(3), is set to zero to indicate that no propagations to the left are necessary. If the ID node is not at the beginning of the line, then the left propagation marker is set one node to the left of the ID node, (ID-1), and atrib(5) is set to zero to indicate a propagation event. Then the LEFTPROP event (event code = 3) is scheduled to be executed once the leading edge of the packet arrives to the (ID-1) node. The time to propagate to the (ID-1) node is stored in the station delay array in the (ID-1) element, stadly(ID-1).

If the ID node is at the end of the line, ID=maxsta, then no propagation to the right is necessary and the right propagation marker must be set to zero. If ID is not equal to maxsta then a propagation event must be declared, atrib(5)=0, and the RIGTPROP event (event code = 4) must be executed when the leading edge of the packet arrives to the (ID+1) node.

It is clear from Figure 3-4 what is happending in the TRANSMIT event. Figure 3-4 shows that in the TRANSMIT event, there is a check to see if a collision had occured as soon as the packet began to transmit. This can happen if the leading edge of a packet arrives to the ID node while the ID node is waiting the interframe spacing. When this occurs, it is said that a collision occured during the interframe spacing and the node state is set to intmrv, which indicates this. If the network status of ID is greater than or equal to two then a packet did arrive to ID during the interframe spacing and the collision subroutine is called.

The final operation performed in TRANSMIT is to schedule the SUCCESS event to be executed after the collision discrimination period has elapsed. Once the collision discrimination period is over the packet controls the network. If the node state does not equal itrans then the ID node will not schedule the SUCCESS event. All the packets are in the itrans state when they enter the TRANSMIT event. It is possible that the node state gets changed to intmrv as described above. As packets leave the TRANSMIT event they are usually in the icolpr state. This indicates that the packet being transmitted is currently in the collision discrimination period.

It has been shown that the TRANSMIT event causes the leading edge of the packet to arrive to the nodes just to the left and right of the ID node. That is, the leading edge of the packet is scheduled to arrive to the nearest nodes by scheduling the LEFTPROP and RIGTPROP events to be executed when the leading edge of the packet arrives to the next node. In addition, the TRANSMIT event handles the situation when a packet arrives to a node while the node is waiting the interframe spacing, and the SUCCESS event (event code = 5) is scheduled to be executed after the collision discrimination period has elapsed.

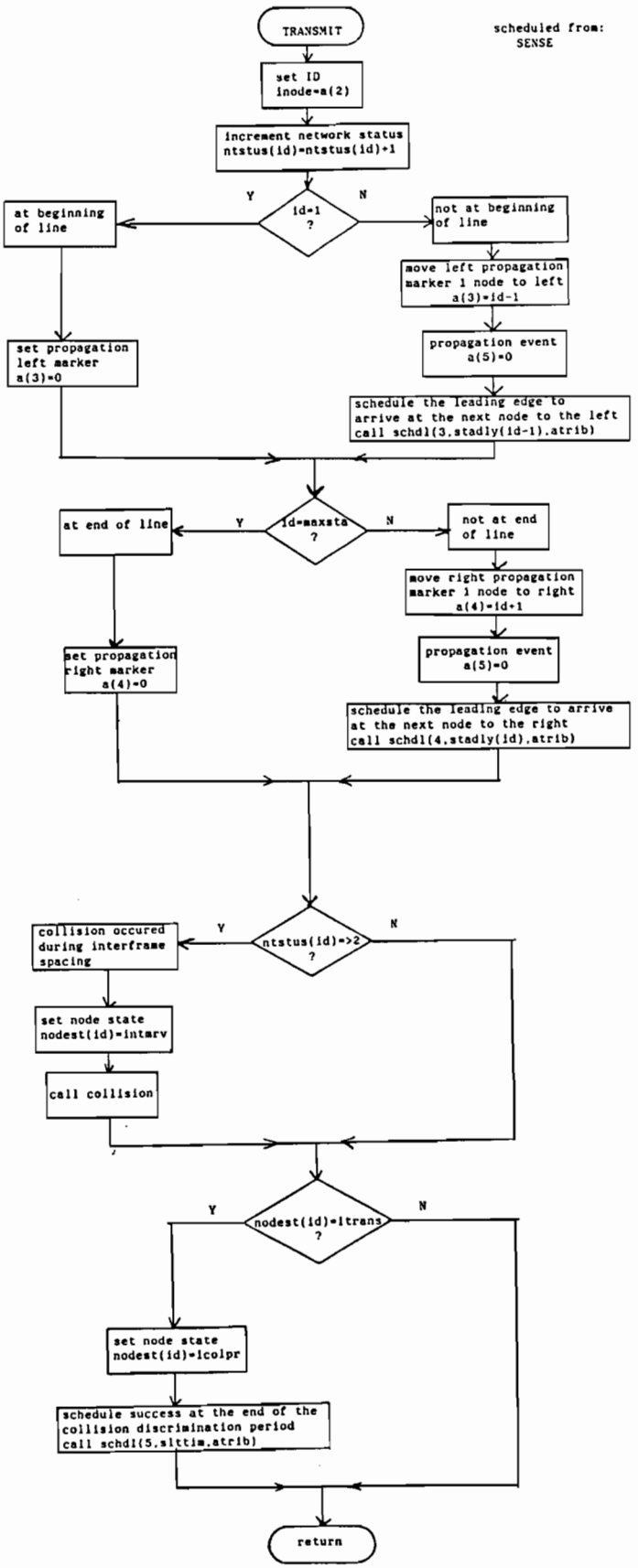


Figure 3-4. Flow diagram of the TRANSMIT event.

In the LEFTPROP event a new variable is created that is used to point to the position to the left of the originating node that the leading edge of the packet has arrived to. The new variable is called poslft. The position left variable is immediately set to the left propagation marker, poslft=atrib(3). The network status is incremented for the poslft node, this causes the poslft node to see the channel as busy, if it was idle before. Next, there is a check to see if incrementing the network status of the poslft node caused it to be greater than or equal to two. If it is and the node state of the poslft node is icolpr, then a collision has occured at the poslft node. If the poslft node was not in the icolpr state, then that node was not transmitting and it does not care that the network status indicates that the channel is undergoing collisions.

If a collision did occur at the poslft node, the collision subroutine must be called, but first the second attribute of the packet must be saved and then set to poslft. This way, the collision subroutine is called for the poslft node, not the node that originated the packet. After calling the collision subroutine, the second attribute is set back to the node that originated the packet.

It is clear from the flow diagram of Figure 3-5 that whether or not a collision occurs at the poslft node there are several other operations performed in the LEFTPROP event. After checking for a collision the left propagation marker is moved one node to the left and poslft is set to the left propagation marker. Then there is a check to see if the new poslft is the first node on the line. If it is, then poslft equals zero and the left propagation marker is set to zero.

If the packet has not propagated all the way down the line to the left then poslft does not equal zero and the leading edge of the packet must be

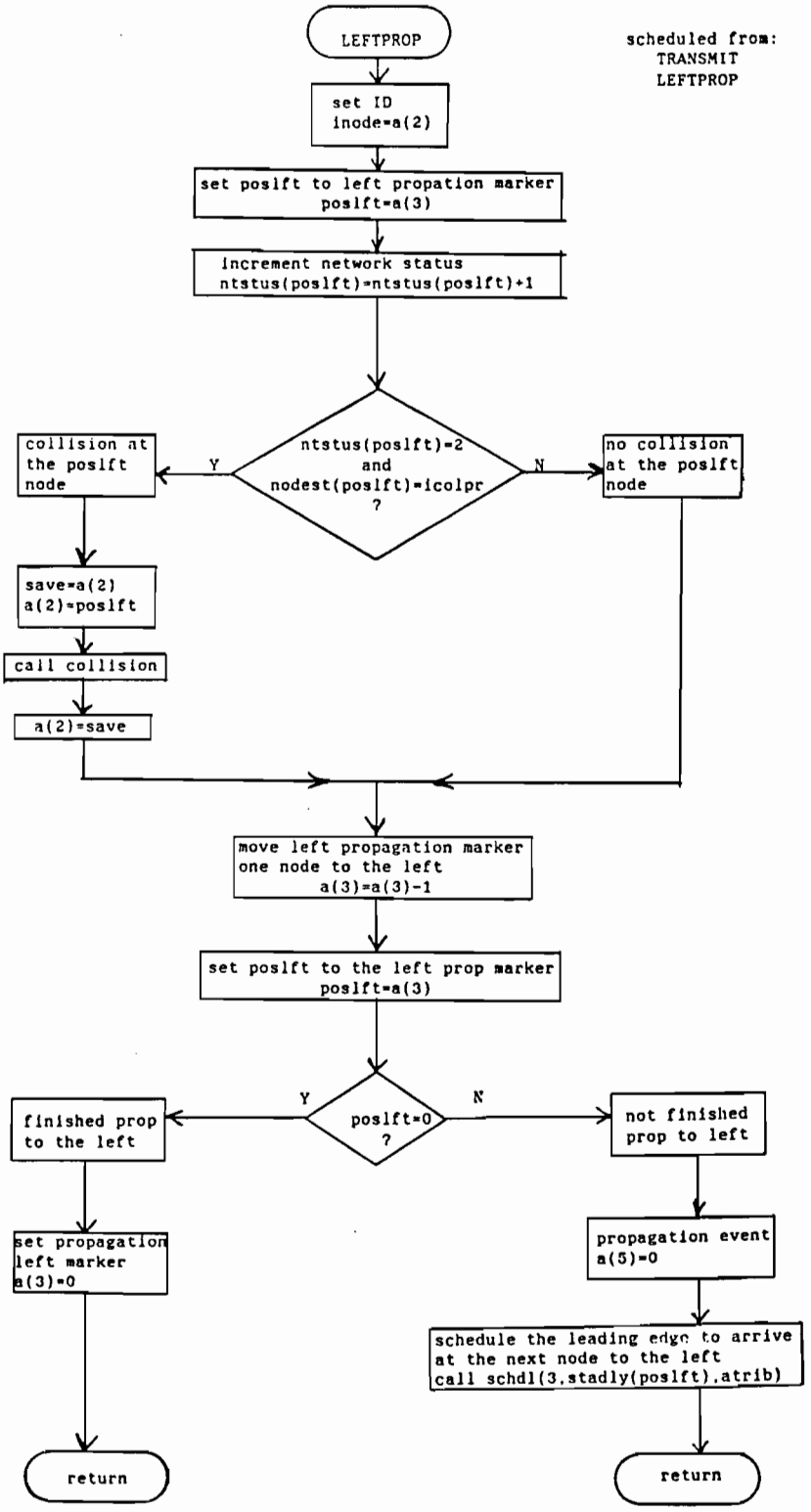


Figure 3-5. Flow diagram of the LEFTPROP event.

scheduled to arrive to the next node to the left. This is done by first setting the fifth attribute to indicate a propagation event, and scheduling the leading edge of the packet to arrive to the next node in the amount of time specified in the station delay array. The LEFTPROP event is being scheduled to be executed in the amount of time required to travel to the next node to the left.

The RIGTPROP event is similar to the LEFTPROP event in that the same operations are performed. In any case, the RIGTPROP event, outlined in the flow diagram of Figure 3-6, will be described here. From Figure 3-6, it is clear that the operations performed to simulate the propagation to the right of the originating node are similar to those done to simulate the propagation to the left. The first thing done is to set the variable posrgt to point to the node that the leading edge of the packet has advanced to. This is done by setting posrgt to the right propagation marker, atrib(4). Then the network status of the posrgt node is incremented. If the posrgt node saw the channel as idle it now sees the channel as busy, and if the posrgt node saw the channel as busy it now sees the channel as undergoing collisions. If the node was not transmitting it will not act on the fact that the channel is undergoing collisions. However, if the node was transmitting (in the collision discrimination node state) then the node would act on the channel condition. This is what is checked for in the decision block that follows the increase in network status. If the network status is equal to two and the node is in the icolpr state, then the node has seen a collision and must call the collision subroutine. If the node was not transmitting and once the collision has been taken care of, the execution proceeds to move the right propagation marker one node to the right.

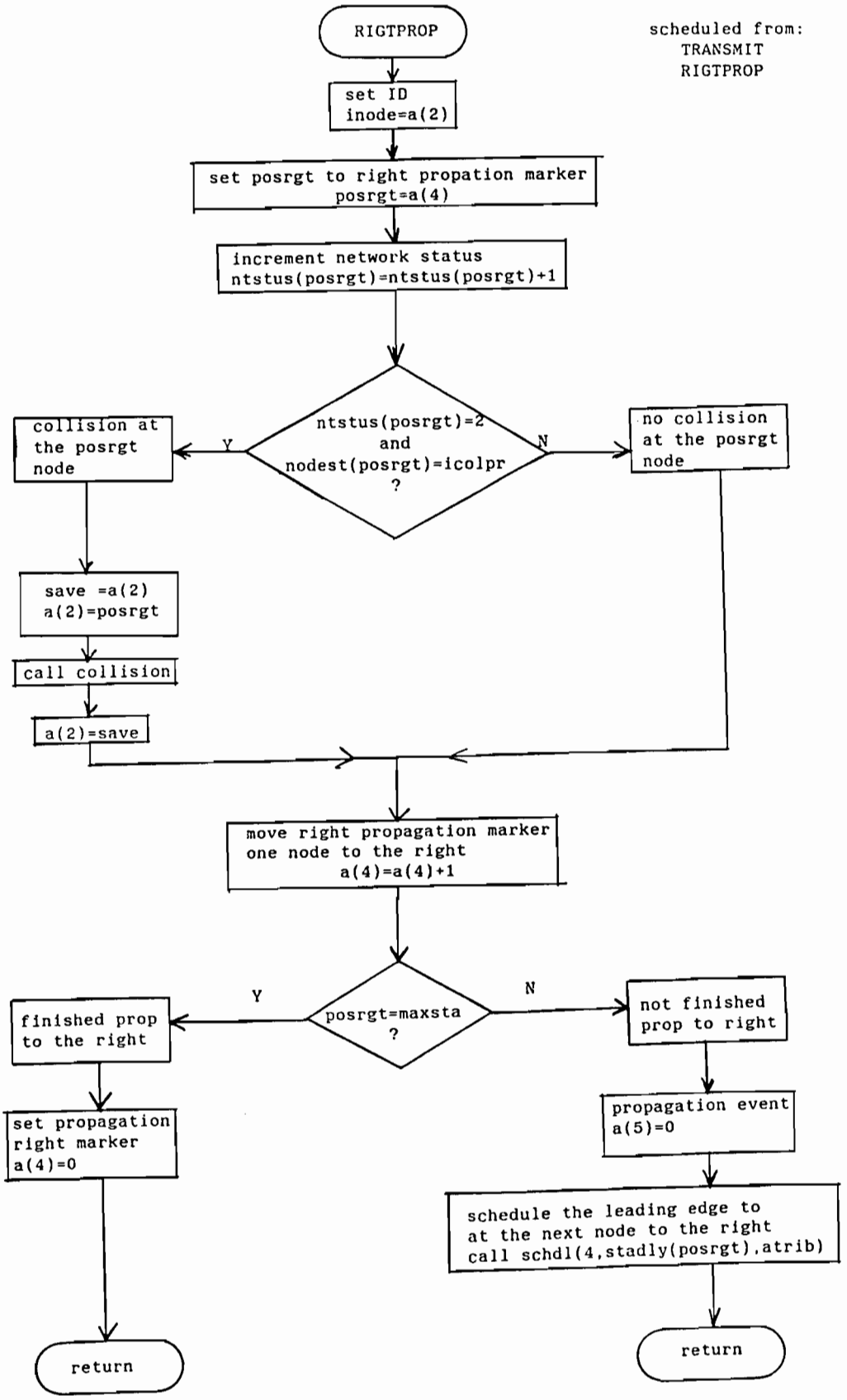


Figure 3-6. Flow diagram of the RIGTPROP event.

It is important to note that when a collision does occur the propagation continues. When a collision occurs the node must stop transmitting, send a jam signal, determine the backoff, wait the backoff, and begin transmitting again by sensing the channel. However, the partial packet that was transmitted before the collision must still propagate to both ends of the line.

To continue with RIGTPROP, after moving the right propagation marker, there is a check to see if the packet has traveled to the end of the line. So, if posrgt is the maxsta node then the leading edge of the packet was just propagated to the last node on the network, and the propagation right marker is set to zero. If posrgt is not equal to maxsta, then the leading edge of the packet must be propagated one node to the right. This is done by first setting atrib(5) to indicate a propagation event and then scheduling the RIGTPROP event to be executed when the packet arrives to the next node to the right. So, the RIGTPROP event, like the LEFTPROP event, is continually executed until the leading edge of the packet has traveled all the way to the end of the line.

Recall that in the TRANSMIT event, the SUCCESS event was scheduled to be executed when the collision discrimination period has elapsed. The SUCCESS event will be executed as long as a collision has not occured. This event is relatively simple, see Figure 3-7. If the SUCCESS event is executed, then the packet has successfully transmitted through the collision discrimination period. The packet can now be transmitted down the line with all the nodes seeing the channel as busy. So, in this event the node state is set to terminate the packet and the ending edge of the packet must be propagated to all the nodes. Recall that the end of the packet was calculated and is in the 10th attribute. So, the ENDTRANS event (event code = 6) is scheduled to be executed after atrib(10) microseconds have elapsed.

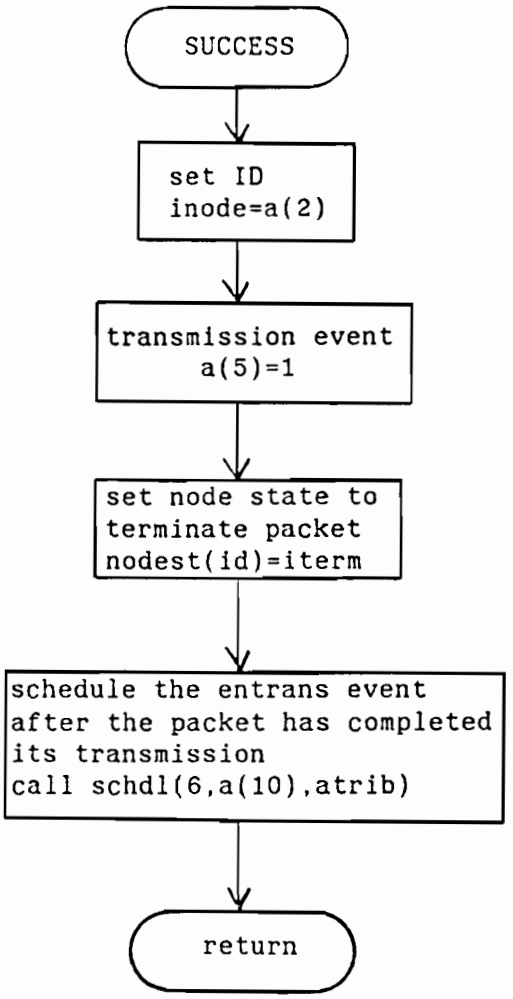


Figure 3-7. Flow diagram for the SUCCESS event.

In the ENDTRANS event the initial propagations of the ending edge of the packet are scheduled to arrive to the next nodes to the left and right of the originating node if the originating node was not at one of the ends of the line. The first thing done is to decrement the network status of the originating node, the ID node. Next, there is a check to see if the ID node is the first node on the line. If it is the end propagation left marker, atrib(7), is set to zero. If the ID node is not the first node on the line the end propagation left marker is set to the next node to the left, and atrib(7) is set to (ID-1). Then atrib(5) is set to zero to indicate a propagation event. Then the LTFINPROP event (event code = 7) is scheduled to be executed when the ending edge of the packet arrives to the (ID-1) node. Whether or not the ID node is at the beginning of the line there is a check to see if the ID node is at the end of the line, see Figure 3-8.

If the ID node is the last node on the network the ID is equal to maxsta. If ID is equal to maxsta the ID node is the last node on the line and the end propagation right marker, strib(8), is set to zero. If ID is not equal to maxsta the ending edge of the packet must be propagated one node to the right. This is done by first setting the right propagation marker to the next node to the right, atrib(8) is set to (ID+1). Then the fifth attribute is set to zero to indicate a propagation event. And finally, the RTFINPROP event (event code = 8) is scheduled to be executed when the ending edge of the packet arrives to the (ID+1) node.

The next operation performed in ENDTRANS is to see if the ID node is in the terminate state. If the ID node is in the terminate state the FREERSC subroutine is called. Then there is a check to see if the ID node is in the defer state. If it is the ID node has a packet ready to begin the process, but nothing is done about it at this point. Then there is a check to see if

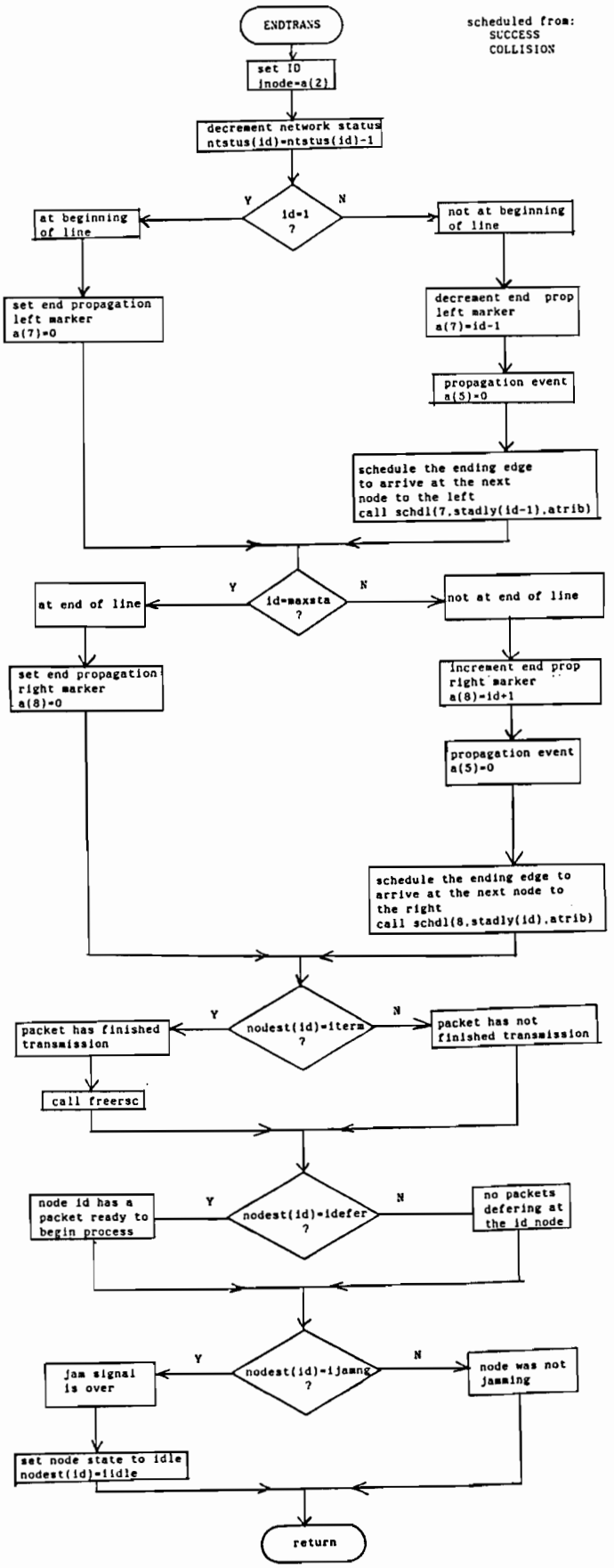


Figure 3-8. Flow diagram of the ENTRANS event.

the node state of the ID node is in the jamming state. If the ID node is in the jamming state the node state is set to idle, and the ENDTRANS event is over.

So, in the ENDTRANS event the simulation of the ending edge of the packet leaving the originating node and the scheduling of it to arrive to the next nodes is done. In the LTFINPROP and RTFINPROP events, the ending edge of the packet is propagated all the way to both ends of the line. The LTFINPROP event is shown in the flow diagram of Figure 3-9.

In LTFINPROP the position left variable, poslft, is set to the end propagation left marker, atrib(7). Then the network status of the poslft node is decremented. Then there is a check to see if the network status of the poslft node is zero and the poslft node is in the deferring state. If both conditions are true, the poslft node has a packet in the defer file, which must be removed and begin the process. So there is a call to the EXITDEFER subroutine which will remove the poslft node's packet from the defer file.

The LTFINPROP event continues by moving the end propagation left marker one node to the left and sets poslft to the end propagation left marker. Then there is a check to see if poslft is the first node on the line. If it is the ending edge of the packet has propagated all the way to the beginning of the line. However, if the poslft node is not the first node on the line a propagation event must be declared and the ending edge of the packet must be scheduled to arrive to the next node to the left, the poslft node. This is done by scheduling the LTFINPROP event to be executed in the amount of time specified in the station delay array. The execution of LTFINPROP will continue until the ending edge of the packet has traveled all the way to the beginning of the line.

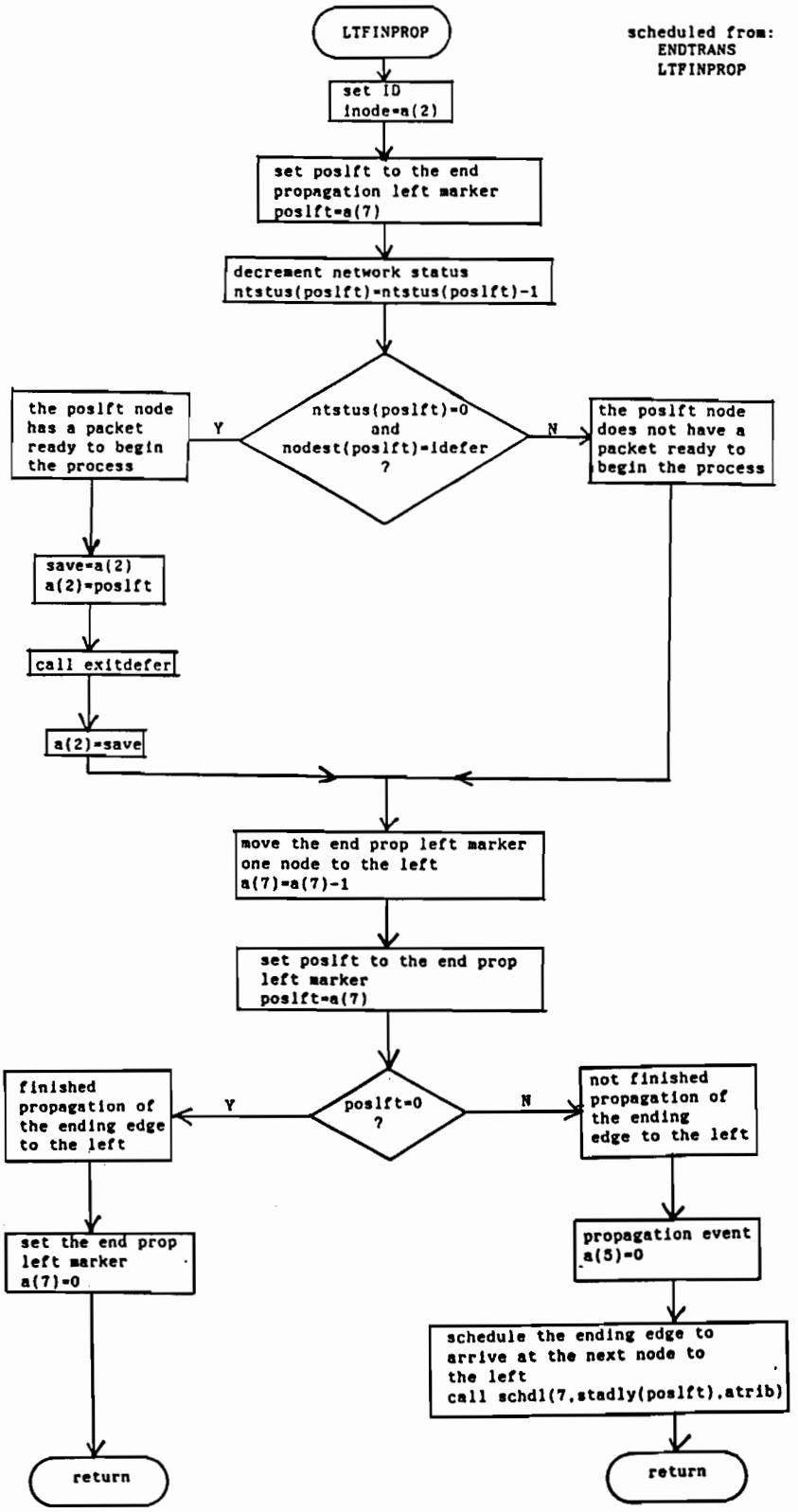


Figure 3-9. Flow diagram of the LTFINPROP event.

The RTFINPROP event is similar to the LTFINPROP event in that the same kinds of operations must be performed when propagating the ending edge of the packet to either end of the line. The RTFINPROP event is shown in the flow diagram of Figure 3-10, which clearly shows the similarity to the LTFINPROP event. The first operation done is to set the position right variable, posrgt, to the end right propagation marker, atrib(8). Then the network status of the posrgt node is decremented, and then a check is made to see if the posrgt node has a packet in the defer file and the channel is idle. If this condition is true the packet is removed by calling the EXITDEFER subroutine. Next, the end right propagation marker is increased by one to point to the next node to be propagated to. A check is made to see if the posrgt node is the maxsta node, if it is the packet has traveled to the end of the line and the end right propagation marker is set to zero. If posrgt is not the maxsta node a propagation event is declared and the RTFINPROP event is scheduled to be executed in the proper amount of time specified in the station delay array. So, like LTFINPROP, the RTFINPROP event will be executed until the ending edge of the packet has traveled all the way to the end of the line.

This completes the discussion of the events used to simulate the CSMA/CD Local Computer Network. In the following section the subroutines, which are listed in Table 3-2, will be described.

3.2.3.2 Description of Subroutines

The subroutines used to simulate the CSMA/CD network will be described here, they include COLLISION, SEARCH, CALC_WAIT_BACKOFF, EXITDEFER,FREERSC, and CHNLECHO. The first three, COLLISION, SEARCH, and CALC_WAIT_BACKOFF, are similar in that they are all executed if a collision occurs. The COLLISION subroutine calls the other two.

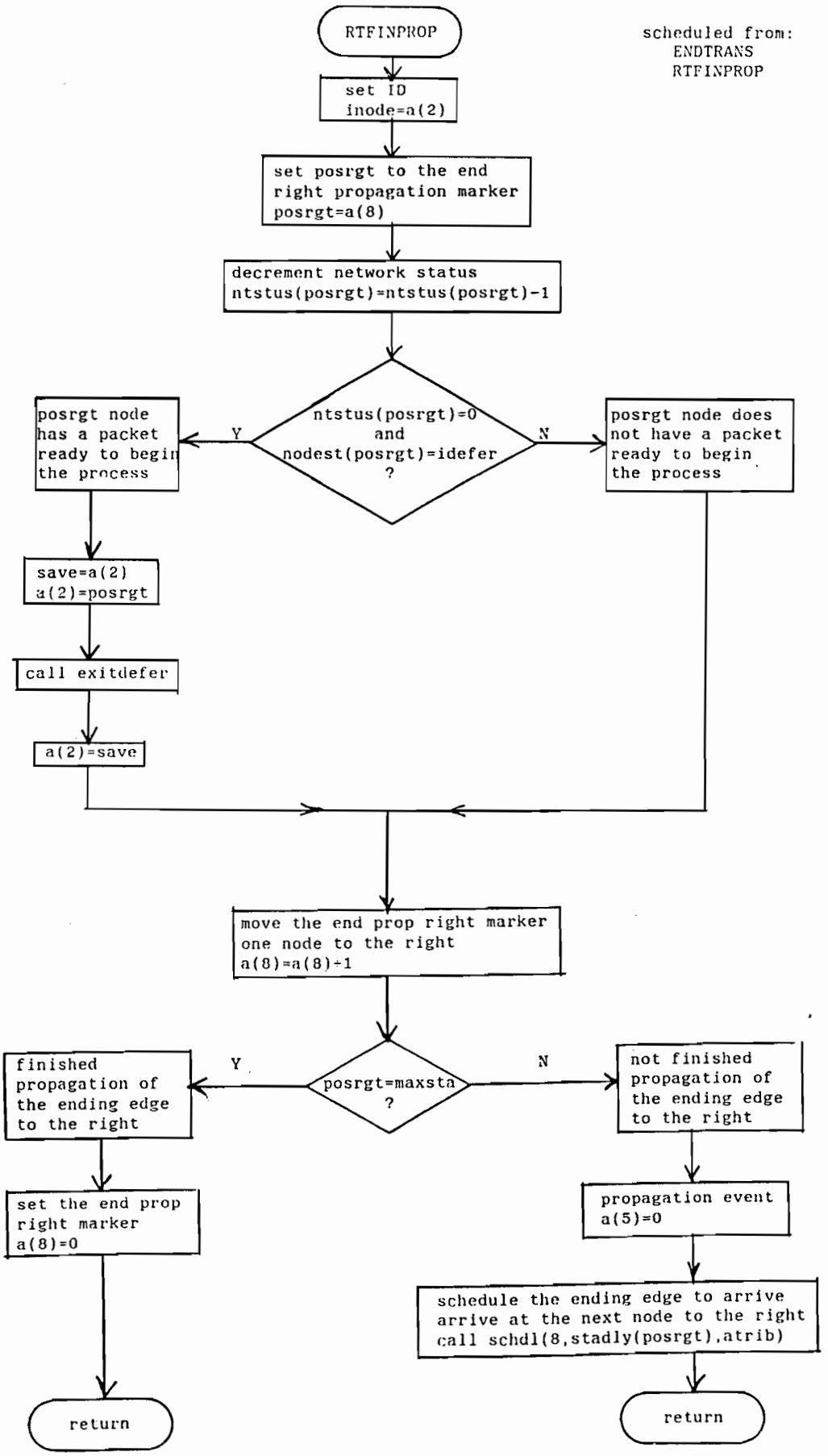


Figure 3-10. Flow diagram of the RTFINPROP event.

The COLLISION subroutine is called from either the TRANSMIT event, the LEFTPROP event, or the RIGTPROP event. If one of the propagation events calls COLLISION the propagation markers must be saved, this is the first thing done in the COLLISION subroutine, as can be seen in Figure 3-11. The next operation done is to see if the node state is intrmv. If the node state is intrmv, a packet arrived to the transmitting node when the node was waiting the interframe spacing and the SEARCH subroutine will not be called. However, if the node state is not intrmv the SEARCH subroutine is called so that the SUCCESS event that was scheduled is removed from the event calendar.

Then the propagation markers are returned, and the sixth attribute is incremented. Recall that the sixth attribute is a counter that keeps track of the number of retransmissions made by the particular packet. Next, the end of the jam signal is scheduled, and the CALC_WAIT_BACKOFF subroutine is called.

The SEARCH subroutine is used to find the SUCCESS event that was scheduled in the TRANSMIT event and remove it from the event calendar. Recall that SUCCESS is executed when a packet successfully transmits through the collision discrimination period. But when a collision occurs the packet did not successfully make it through the collision discrimination period, so the SUCCESS event should not be executed and it must be removed from the event calendar.

The SEARCH subroutine uses many of the SLAM functions and subroutines described in Chapter 2. The first thing done is to set the pointer "next" to point at the first entry in the event calendar, file NCLNR. This is done using the MMFE function. Then the entry pointed to by "next" is copied into the attribute buffer, and a check is made to see if this entry has its second attribute equal to ID and its fifth attribute equal to one. If this is true then this entry is the SUCCESS event that was due to occur and must be re-

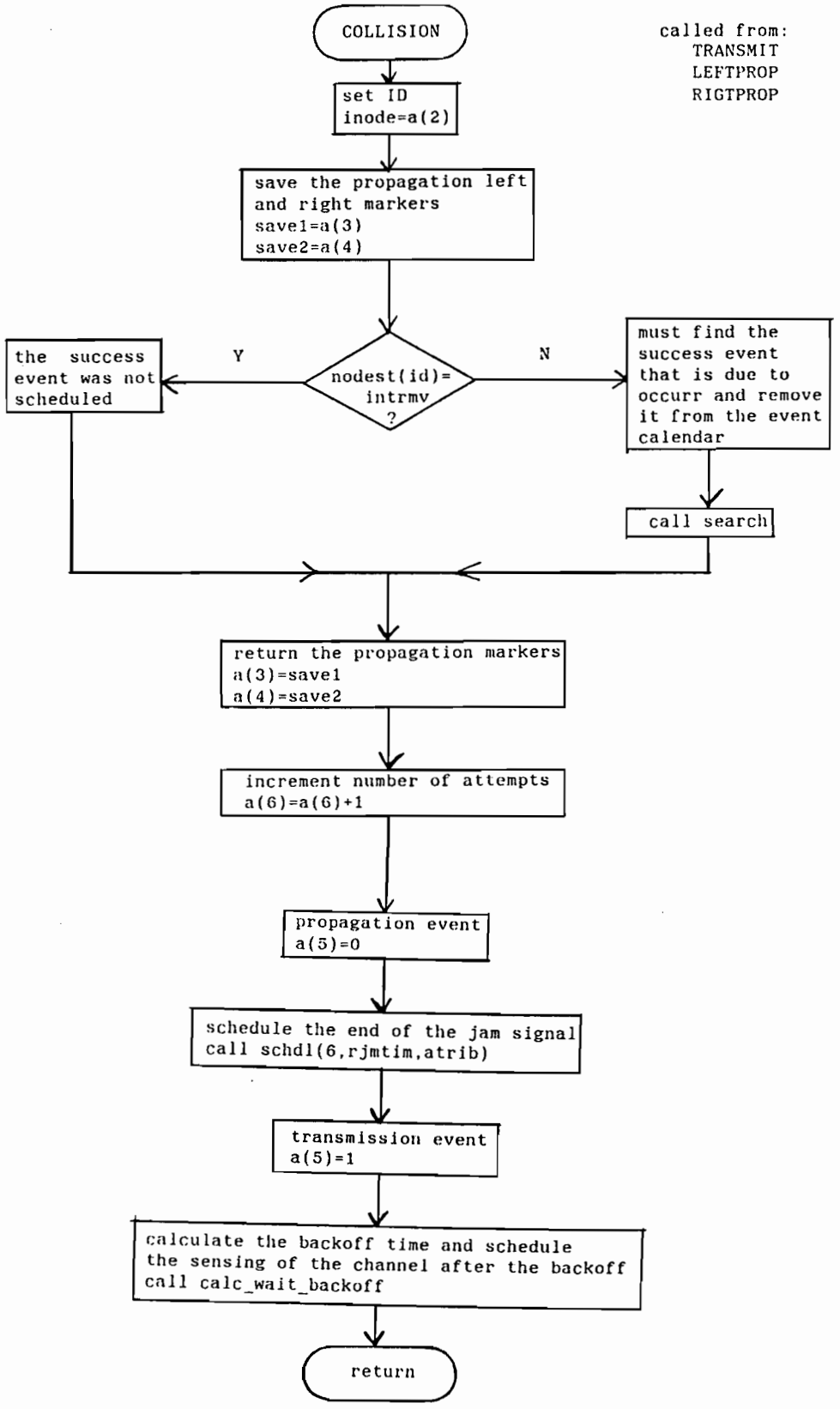


Figure 3-11. Flow diagram of the COLLISION subroutine.

moved. The event will be removed using the RMOVE subroutine. If the condition is not met the pointer "next" must be incremented using the NSUCR function and execution returns to copy this event into the attribute buffer, see Figure 3-12. The operations are continued until the correct event is removed from the event calendar.

In the CALC_WAIT_BACKOFF subroutine there is a check to see if too many collisions have occured. If there have been too many then atrib(6) is equal to mxcoll, (typically 16 [5]) and the packet must be terminated. If atrib(6) is not equal to mxcoll execution continues by checking to see if atrib(6) is greater than or equal to 8. If atrib(6) is greater than or equal to 8 then there have been 8 or more retransmission attempts and the upper limit of the uniform distribution will be truncated. If atrib(6) is less than 8, the upper limit is calculated as shown in Figure 3-13. Once the upper limit of the uniform distribution has been determined a sample from the uniform distribution will be taken. This sample is then converted to an integer and multiplied by the slot time. This final number is the amount of time that will be waited before sensing the channel again. The last thing done is to schedule a sensing of the channel in the amount of time specified by the backoff.

From the above it was shown that the three subroutines, COLLISION, SEARCH, and CALC_WAIT_BACKOFF are all executed when a collision occurs. They make up the total process that must be executed when a collision occurs. The final three subroutines, EXITDEFER, FREERSC, and CHNLECHO will be described next.

Recall that the EXITDEFER subroutine is called from the LTFINPROP and RTFINPROP events when the ending edge of a packet arrives to a node that has a packet waiting in the defer file. In the EXITDEFER subroutine, shown in Figure 3-14, the packet is removed from the defer file and immediately senses

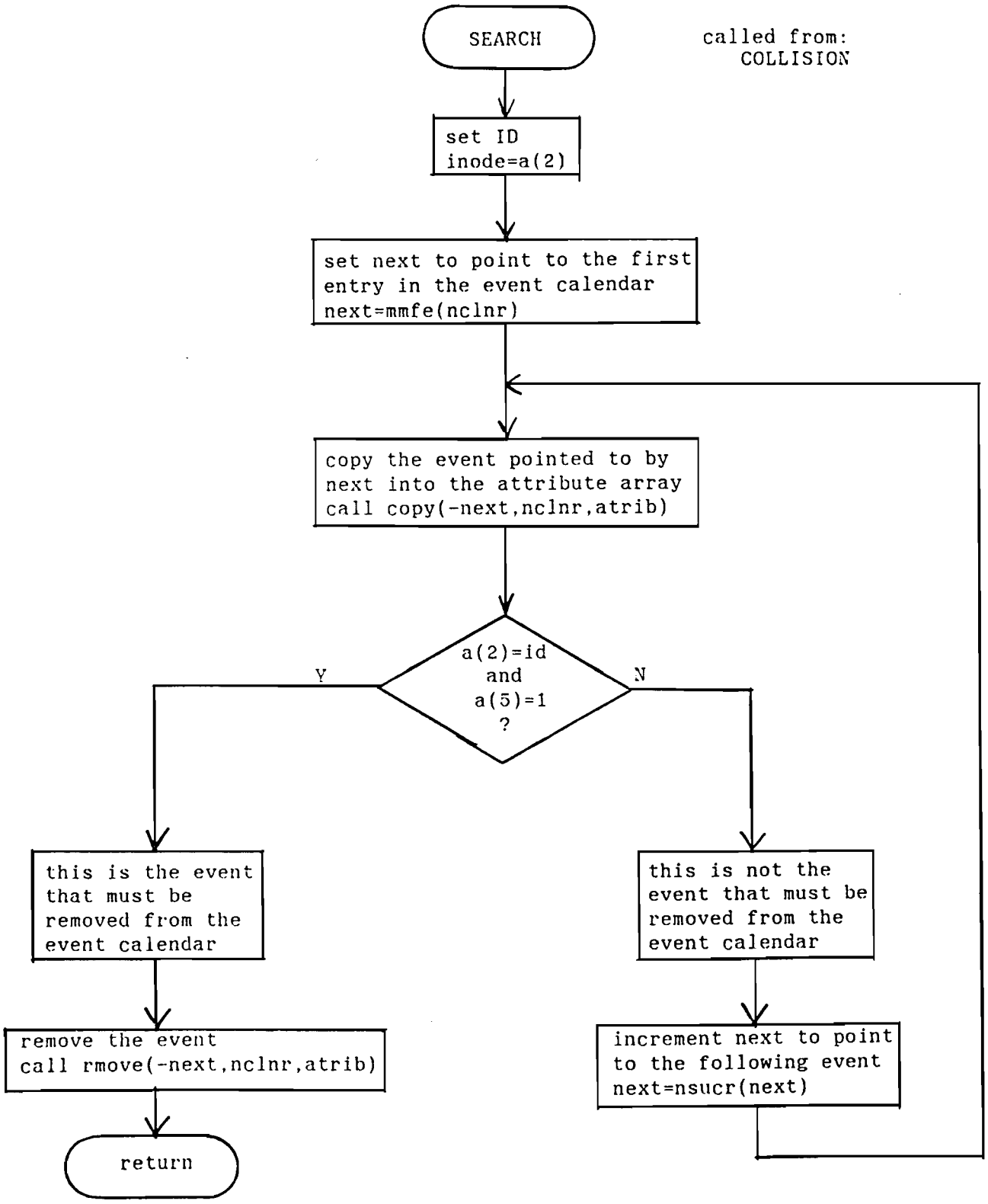


Figure 3-12. Flow diagram of the SEARCH subroutine.

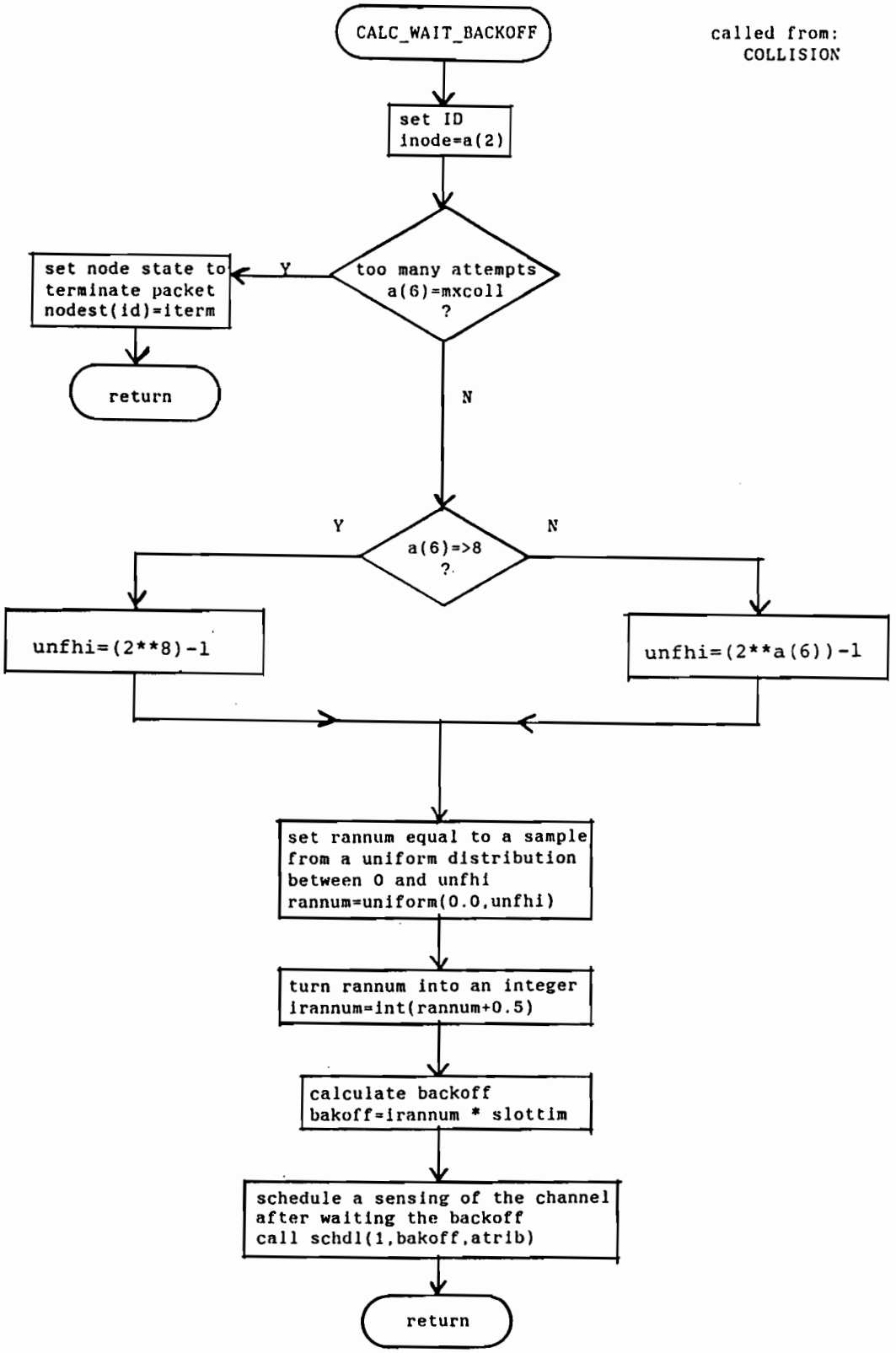


Figure 3-13. Flow diagram of the CALC_WAIT_BACKOFF subroutine.

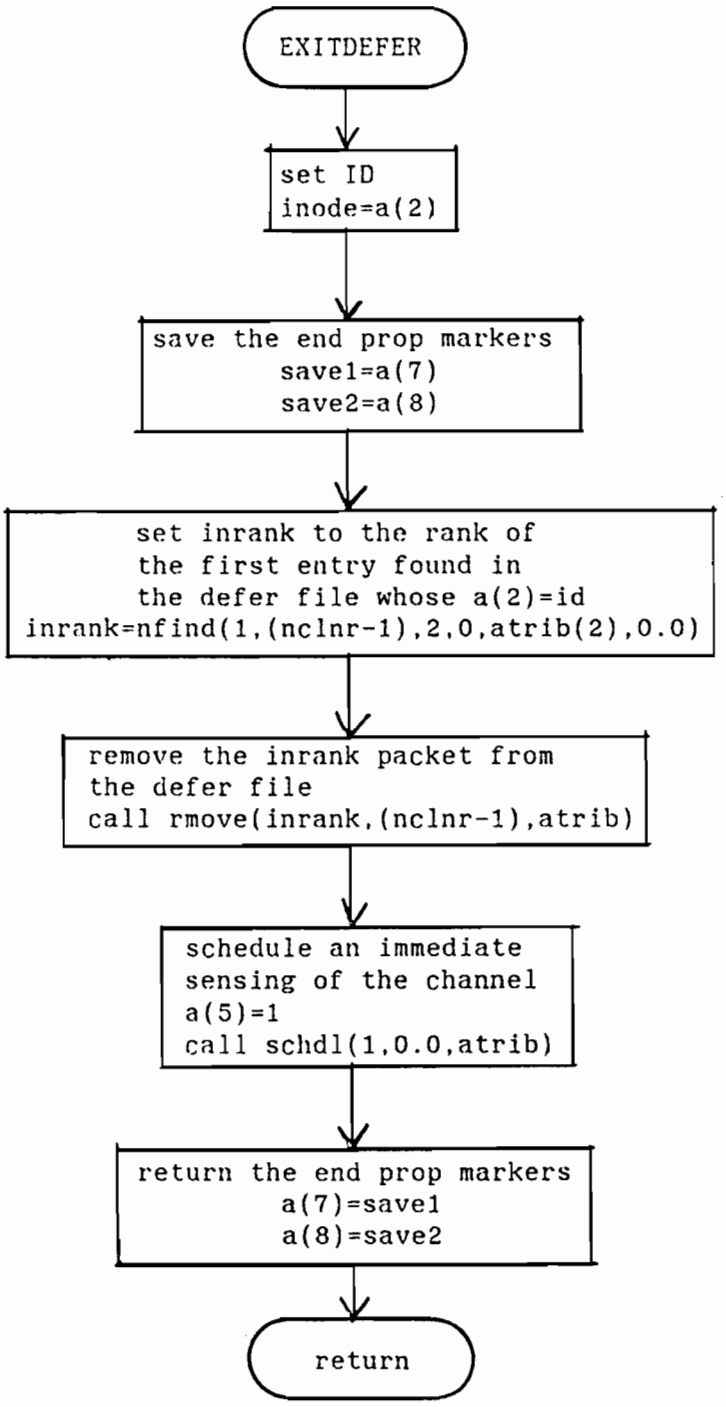


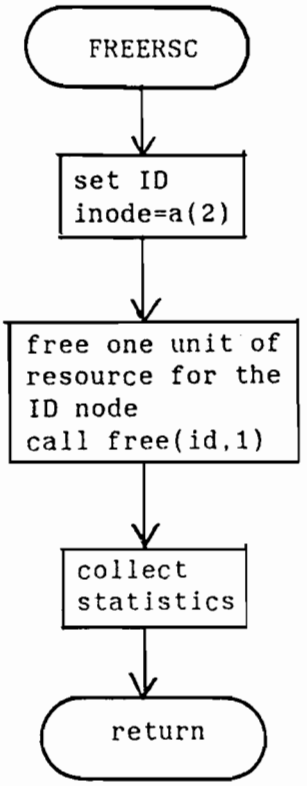
Figure 3-14. Flow diagram of the EXITDEFER subroutine.

the channel. The first thing done is to save the end propagation markers. Then the packet that must be removed from the defer file is found by using the SLAM function NFIND. Recall from Chapter 2 that NFIND is used to find an entry in a file that bears a relationship to an attribute. The "inrank" variable is set to the rank of the entry in the defer file whose second attribute is exactly equal to the ID node number. This entry is the packet that must be removed using the RMOVE subroutine, recall that the defer file is numbered (NCLNR-1). Next, an immediate sensing of the channel is scheduled and the end propagation markers are returned.

The FREERSC subroutine is discussed here and is perhaps the simplest to describe. All that is done in FREERSC that has significance to modeling is that the FREE subroutine, provided by SLAM, is used to free one unit of resource for the packet that has finished its transmission or is being discarded due to excessive collisions. This allows the next packet, if there is one, to leave its node queue, the AWAIT block, and begin the process. Much of the statistic collection is also done in FREERSC, see Figure 3-15.

The CHNLECHO subroutine does not perform any of the modeling, it is used to check the intermediate results of the simulation. The CHNLECHO subroutine will print out the event or subroutine that called it, the current time, the origin node, the network status array, and the node state array.

This completes the description of the events and subroutines used to simulate the CSMA/CD network. The following three subsections will complete the description of the CSMA/CD simulation model. They include a discussion on initializing the simulation, the output from the simulation, and how to change the system configuration. The final section of this chapter will be a discussion of the validation of the simulation.



called from:
ENDTRANS

Figure 3-15. Flow diagram of the FREERSC subroutine.

3.2.4 Initializing the Simulation

As described in Chapter 2 section 2.2.1, the INTLC subroutine can be used to initialize the variables used to simulate the CSMA/CD network. The INTLC subroutine is executed by the SLAM processor before the simulation begins. This way all the variables that must be initialized are initialized before any packets are generated. In addition to initializing variables, some variables are set and remain unchanged throughout the simulation. The listing of the INTLC subroutine is in Appendix A.3. In this section, there is a description of the variables that are initialized and set.

The excoll and icoll arrays are initialized to zero. These arrays contain the number of packets lost to excessive collisions and the total number of collisions experienced by each node, respectively. The frstat and icnt arrays are also initialized to zero, these arrays contain the number of packets successful on the first attempt to access the network and the total number of packets transmitted from each node, respectively. The timegd array is initialized to zero, this array contains the time spent sending packets successfully from each node. The timegd divided by the total time equals the throughput. The node state array, nodest, is initialized to 5, which is the idle state. The network status array, ntstus, is initialized to zero to indicate that the channel is initially idle. The attempts array is initialized to zero, recall that the elements in this array contain the number of packets successful after that elements number of attempts. The station delay array, stadly, is set in the INTLC subroutine. The station delay array specifies the spacing between network nodes. This spacing is in time and can be the same for all nodes or different. The bitsbd and bitsgd counters are initialized to zero, these counters keep track of the number of bits that were unsuccessful and successful, respectively.

3.2.5 Output From the Simulation

The output from the simulation consists of two parts. The first is the standard SLAM Summary Report and the second is the output from the OTPUT subroutine. In this section, the two output types will be discussed.

The SLAM Summary Report is the standard output of SLAM models. It includes the statistics for variables based on observation. These variables are those described in section 3.2.2 which are defined in the Network Model using the STAT statements. The SLAM Summary Report also gives file statistics. This is the statistics for the node queues, the AWAIT blocks in the Network Model. Included in the file statistics are the statistics for the channel queue, the defer file, and the event calendar. There is information given for the service activity statistics and the resource statistics. For an example of a SLAM Summary Report output from the CSMA/CD simulation, see Figure 3-16.

The other output type is that which is the result of executing the OTPUT subroutine. Once the end of the simulation is reached, the SLAM processor executes the OTPUT subroutine and the particular output that the analyst has chosen to have printed out will be written. The OTPUT subroutine has been included so that the simulation results of some specific performance indicators could be reported. These performance indicators include throughput, number of collisions, number of packets successful on the first attempt to access the network, number of packets successful after a given number of attempts, number of packets transmitted, number of packets discarded due to excessive collisions, number of successfully transmitted bits, and the number of bits that were unsuccessful. The OTPUT subroutine is listed in Appendix A.4, and a sample output is shown in Figure 3-17.

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT ETHERNET BY TISL

DATE 8/20/1984 RUN NUMBER 1 OF 1

CURRENT TIME 0.1000E+08
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
NODE 1 SYS. TIME	0.2289E+04	0.1838E+04	0.8030E+00	0.1403E+04	0.1693E+05	734
NODE 2 SYS. TIME	0.2257E+04	0.1640E+04	0.7265E+00	0.1403E+04	0.2053E+05	766
NODE 3 SYS. TIME	0.2242E+04	0.1735E+04	0.7737E+00	0.1403E+04	0.1681E+05	707
NODE 4 SYS. TIME	0.2278E+04	0.1584E+04	0.6953E+00	0.1403E+04	0.1354E+05	799
NODE 5 SYS. TIME	0.2132E+04	0.1389E+04	0.6516E+00	0.1403E+04	0.1686E+05	656
QUEUEING DELAY	0.2490E+03	0.8819E+03	0.3542E+01	0.0000E+00	0.1040E+05	3663
ACCESS DELAY	0.6020E+03	0.1404E+04	0.2333E+01	0.9500E+01	0.1679E+05	3662
CHANNEL DELAY	0.1393E+04	0.0000E+00	0.0000E+00	0.1393E+04	0.1394E+04	3658
TOTAL SYS DELAY	0.2243E+04	0.1647E+04	0.7345E+00	0.1403E+04	0.2053E+05	3662

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	AWAIT	0.0165	0.1341	2	0	225.1501
2	AWAIT	0.0216	0.1753	4	0	282.0020
3	AWAIT	0.0143	0.1214	2	0	202.5833
4	AWAIT	0.0255	0.1807	3	0	318.6508
5	AWAIT	0.0133	0.1221	3	0	202.2252
22	QUEUE	0.0000	0.0000	0	0	0.0000
24		0.1934	0.4923	4	0	457.4522
25	CALENDAR	5.5473	0.5680	20	6	544.1663

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
0	CHNL QUEUE	1	0.0000	0.0000	0	0.0000	25017.0000	0.0000	

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTILIZATION	CURRENT UTILIZATION
1	PACK1	1	0.1515	0.3585	1	0
2	PACK2	1	0.1514	0.3584	1	1
3	PACK3	1	0.1442	0.3513	1	0
4	PACK4	1	0.1566	0.3634	1	0
5	PACK5	1	0.1266	0.3325	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	PACK1	1	0.8485	0	1
2	PACK2	0	0.8486	0	1
3	PACK3	1	0.8558	0	1
4	PACK4	1	0.8434	0	1
5	PACK5	1	0.8734	0	1

Figure 3-16. Sample SLAM Summary Report from the CSMA/CD simulation.

```
TIME SPENT SENDING GOOD PACKETS FROM STATION 1 = 1021258.38   THROUGHPUT = 0.102
TIME SPENT SENDING GOOD PACKETS FROM STATION 2 = 1067234.50   THROUGHPUT = 0.107
TIME SPENT SENDING GOOD PACKETS FROM STATION 3 =  982258.69   THROUGHPUT = 0.098
TIME SPENT SENDING GOOD PACKETS FROM STATION 4 = 1113218.50   THROUGHPUT = 0.111
TIME SPENT SENDING GOOD PACKETS FROM STATION 5 =  912579.94   THROUGHPUT = 0.091

THE TOTAL TIME SPENT WITH GOOD PACKETS = 5096550.00     TOTAL THROUGHPUT = 0.510


TOTAL # OF COLLISIONS THAT OCCURED AT STATION 1 = 697
TOTAL # OF COLLISIONS THAT OCCURED AT STATION 2 = 741
TOTAL # OF COLLISIONS THAT OCCURED AT STATION 3 = 649
TOTAL # OF COLLISIONS THAT OCCURED AT STATION 4 = 682
TOTAL # OF COLLISIONS THAT OCCURED AT STATION 5 = 508

TOTAL # OF COLLISIONS = 3277      COLLISIONS PER MILLISECOND = 0.3277


# OF PACKETS SUCCESSFUL ON THE FIRST ATTEMPT TO ACCESS NET FROM 1 = 301
# OF PACKETS SUCCESSFUL ON THE FIRST ATTEMPT TO ACCESS NET FROM 2 = 306
# OF PACKETS SUCCESSFUL ON THE FIRST ATTEMPT TO ACCESS NET FROM 3 = 279
# OF PACKETS SUCCESSFUL ON THE FIRST ATTEMPT TO ACCESS NET FROM 4 = 309
# OF PACKETS SUCCESSFUL ON THE FIRST ATTEMPT TO ACCESS NET FROM 5 = 252

       TOTAL # OF 1st ACCESS = 1447        % OF 1st ACCESS = 39.5139


# OF PACKETS SUCCESSFUL ON THE  1st  ATTEMPT = 2797        PERCENTAGE = 76.379
# OF PACKETS SUCCESSFUL ON THE  2nd  ATTEMPT =  169        PERCENTAGE =  4.615
# OF PACKETS SUCCESSFUL ON THE  3rd  ATTEMPT =   31        PERCENTAGE =  0.847
# OF PACKETS SUCCESSFUL ON THE  4th  ATTEMPT =  222        PERCENTAGE =  6.062
# OF PACKETS SUCCESSFUL ON THE  5th  ATTEMPT =  214        PERCENTAGE =  5.844
# OF PACKETS SUCCESSFUL ON THE  6th  ATTEMPT =   87        PERCENTAGE =  2.376
# OF PACKETS SUCCESSFUL ON THE  7th  ATTEMPT =   57        PERCENTAGE =  1.557
# OF PACKETS SUCCESSFUL ON THE  8th  ATTEMPT =   35        PERCENTAGE =  0.956
# OF PACKETS SUCCESSFUL ON THE  9th  ATTEMPT =   16        PERCENTAGE =  0.437
# OF PACKETS SUCCESSFUL ON THE 10th  ATTEMPT =   12        PERCENTAGE =  0.328
# OF PACKETS SUCCESSFUL ON THE 11th  ATTEMPT =    9        PERCENTAGE =  0.246
# OF PACKETS SUCCESSFUL ON THE 12th  ATTEMPT =    3        PERCENTAGE =  0.082
# OF PACKETS SUCCESSFUL ON THE 13th  ATTEMPT =    2        PERCENTAGE =  0.055
# OF PACKETS SUCCESSFUL ON THE 14th  ATTEMPT =    1        PERCENTAGE =  0.027
# OF PACKETS SUCCESSFUL ON THE 15th  ATTEMPT =    3        PERCENTAGE =  0.082
# OF PACKETS SUCCESSFUL ON THE 16th  ATTEMPT =    0        PERCENTAGE =  0.000

          % SUCCESSFUL ON 1st ATTEMPT =. 76.3790


TOTAL # OF PACKETS TRANSMITTED FROM STATION 1 = 734
TOTAL # OF PACKETS TRANSMITTED FROM STATION 2 = 766
TOTAL # OF PACKETS TRANSMITTED FROM STATION 3 = 707
TOTAL # OF PACKETS TRANSMITTED FROM STATION 4 = 799
TOTAL # OF PACKETS TRANSMITTED FROM STATION 5 = 656

TOTAL # OF PACKETS TRANSMITTED =          3662



# OF DISCARDED PACKETS FROM 1 = 1
# OF DISCARDED PACKETS FROM 2 = 0
# OF DISCARDED PACKETS FROM 3 = 2
# OF DISCARDED PACKETS FROM 4 = 0
# OF DISCARDED PACKETS FROM 5 = 1

TOTAL # OF DISCARDED PACKETS = 4      % OF PACKETS DISCARDED = 0.1092


  TOTAL # OF SUCCESSFUL BITS TRANSMITTED = 14983168

TOTAL # OF UNSUCCESSFUL (DISCARDED) BITS = 16384
```

Figure 3-17. Sample output of the OTPUT subroutine.

3.2.6 Changing the System Configuration

The analyst will most likely want to change the system configuration at some time. This can be done very easily by modifying the Network Model and the PARAMS file. The PARAMS file is used to avoid typing the declaration statements at the top of each event and subroutine. In it there are several common blocks and parameter statements. The parameter statements are provided by FORTRAN 77 and are a way of setting up constants. The PARAMS file is listed in Appendix A.5. In this section, a brief description of how the system configuration is modified will be given.

To change many of the system parameters requires that the variable be modified in the PARAMS file. Such things as the jam signal, the slot time, the interframe spacing, and the maximum number of collisions can be changed by modifing them in the PARAMS file. To change the number of nodes being simulated requires that the maxsta parameter in the PARAMS file be changed and that the Network Model of Appendix A.1 be changed. In the Network Model the analyst must remove the semicolon in the first column of a node. This will turn that particular node on. The line capacity is changed by changing the mean interarrival time in the CREATE blocks of the Network Model. Recall from equation (3-3) that the mean interarrival time is a function of the line capacity. Also the packet length in microseconds, atrib(9), must be modified if the line capacity is changed. Recall that the packet length in seconds equals the packet length in bits divided by the line capcity.

3.3 Validation of the Simulation Model

In this section, an experiment which duplicates Shoch's measurements [6, 7, 8] will be presented. This experiment serves as a way to validate the simulation.

In Shoch's work, the Ethernet has the following configuration [6]:

- line capacity: 2.94M bits/second,
- bus length: 550 meters (propagation delay = 2.75 microseconds),
- slot time: round-trip bus delay = 5.5 microseconds
- backoff algorithm: binary exponential backoff, truncated at 2^8,
- jam time after a collision = 32 bits = 10.8844 microseconds,
- packet length: 4096 bits per packet,
- packet arrival rate: 10% per host, for 5 to 15 hosts.

From the above configuration and using equation (3-3), the mean packet interarrival time can be calculated as 13,931.97265625 seconds per packet for Poisson arrivals. The simulation was run under the above configuration. The results are compared with Shoch's measurements. Figure 3-18 presents the system throughput under various traffic loads and a packet length of 4096 bits per packet. In Table 3-3, the fairness of access among network nodes are compared. These results show that the simulation model is able to closely simulate the behavior of a real system.

The simulation can produce the average system delay, which Shoch was not able to measure. Figure 3-19 presents the simulated average system delay with a comparison to the simulation results of Hughes and Li [9]. A comparison of the simulated average system delay and the delay reported by Acampora et al [10], shows good agreement.

In this chapter the CSMA/CD simulation model has been described. The programs are listed in Appendix A. The validation of the model was presented, and it was seen that the model closely simulates a real system. The simulation can be used to do performance studies of Ethernet-like networks. The simulation can also be used to do voice/data performance studies of Ethernet-like Local Area Networks, and that is the subject of Chapter 4.

Total Offered Load (%)	Shoch's Measurement			Simulated Result		
	Total Util.(%)	Avg. Util.(%)	Range	Total Util.(%)	Avg. Util.(%)	Range
70	70	10.0	10-10	71.0	10.1	9.7-10.5
80	80	10.0	10-10	79.6	10.0	8.9-10.5
90	90	10.0	10-10	86.5	9.6	9.0-10.5
100	94	9.4	9.3-9.5	91.2	9.1	8.5-9.8
120	96	8.0	7.8-8.1	94.6	7.9	7.5-8.5
150	96	6.4	5.8-7.1	96.1	6.4	6.1-7.0

Table 3-3. Comparison of fairness of access.

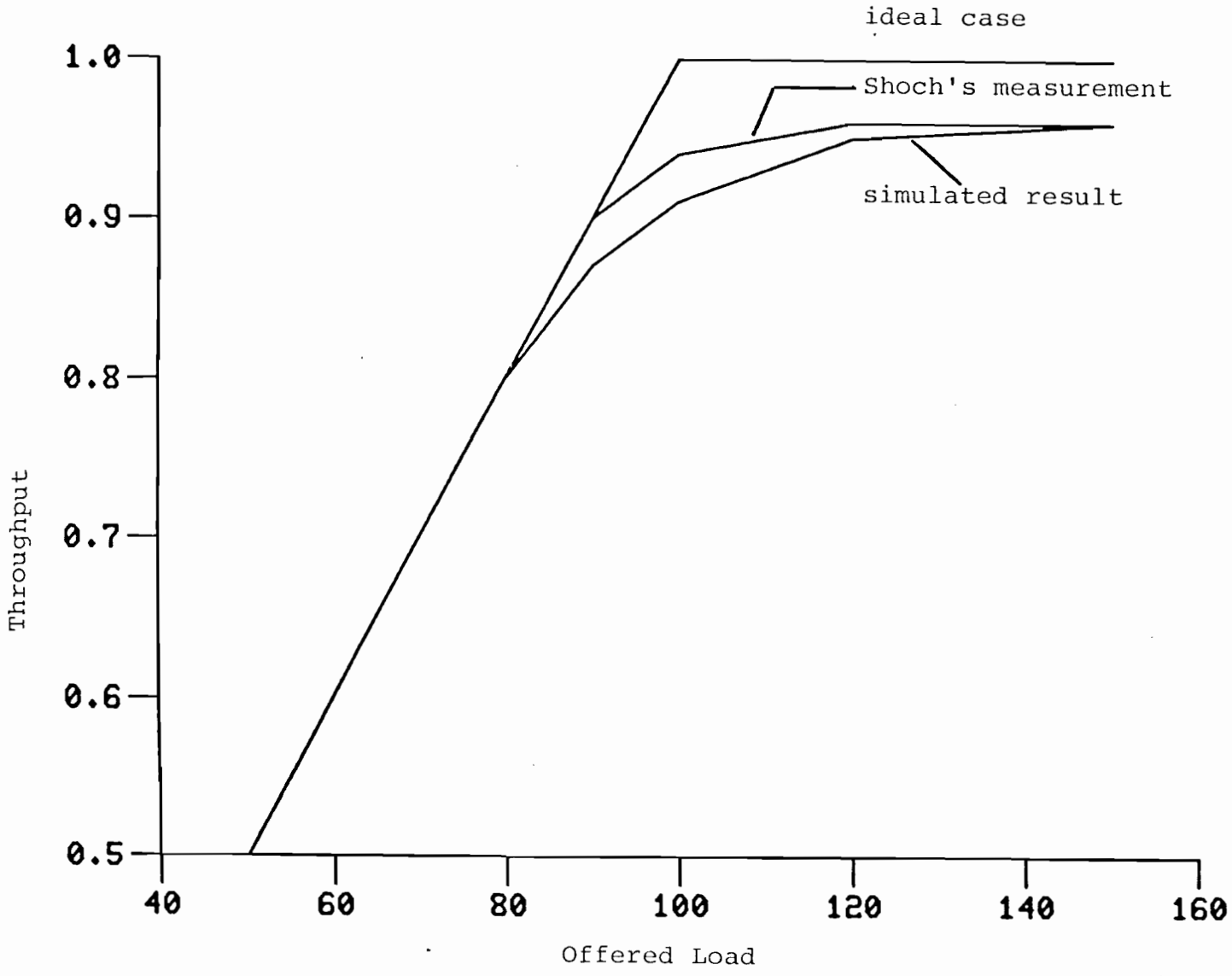


Figure 3-18. Comparison of the simulated throughput to Shoch's measurements, under various traffic loads and packet length of 4096 bits.

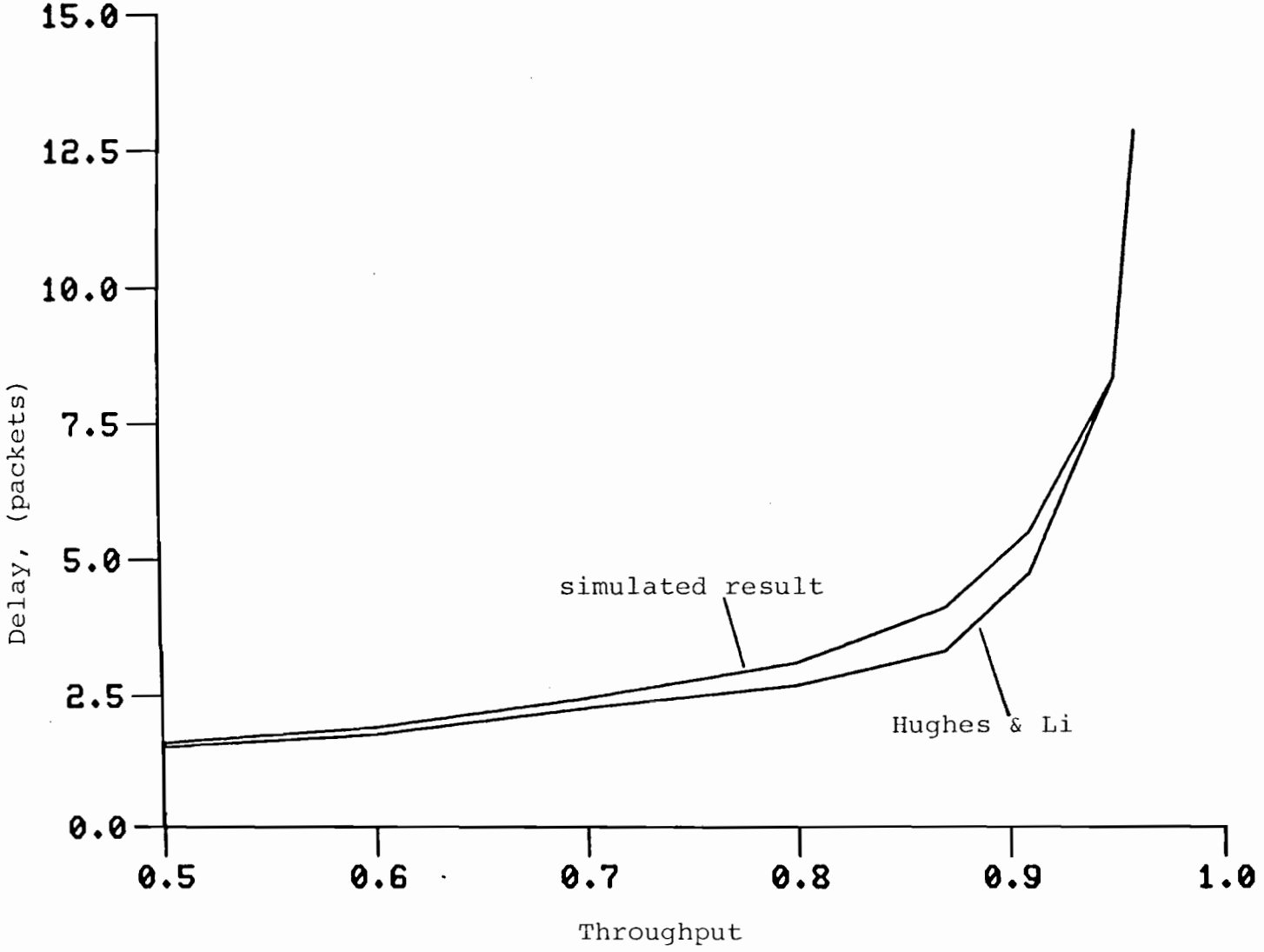


Figure 3-19. Comparison of the simulated average system delay to the delay reported by Hughes and Li, under various traffic loads and a packet length of 4096 bits.

4.0 MULTIRATE VOICE CODING FOR LOAD CONTROL ON CSMA/CD NETWORKS

In Chapter 3 the simulation of a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) network was discussed. It was shown that the simulation model accurately predicts the performance of a real system. This simulation model can now be used to perform simulation studies of voice/data networks. In this chapter the simulation of a voice/data network in which the voice coding rate changes with the load will be discussed.

There have been previous studies of combined voice/data Local Area Networks [12, 13, 14]. The study done by DeTreville [12] was a comparison of the CSMA/CD type of network with a token bus. In that study the token bus was found to perform somewhat better. The Nutt and Bayer study [13] gives a comparison of various backoff algorithms appropriate for voice. In the study by Musser, et al., [14] the virtual token protocol (GBRAM) and the CSMA/CD protocol were compared, in addition, listener tests were performed which confirmed that two percent of the voice packets could be lost and the voice quality would still be acceptable to the user. The results presented in this study agree with the results given in [12, 13, 14]. In this study the applicability in using a multirate voice coding scheme for load control will be considered.

The simulation of voice on a Local Area Network is characterized by periodic arrivals. With periodic voice packet arrivals the voice source is in the talk-spurt mode all the time. The voice packet is characterized by its generation period. The generation period is the packet length in bits divided by the voice coding rate in bits-per-second, see Equation (4-1).

$$G = P/R \qquad (4\text{-}1)$$

where,

G = generation period in seconds,

P = packet length in bits

R = voice coding rate in bits-per-second

After every generation period seconds a new voice packet has been created. There is a continuous flow of voice packets from the source, however, the time to transmit over the line is much less than the generation period. The time required to transmit the packet is the packet length in bits divided by the line capacity in bits-per-second. If a packet has not successfully transmitted over the network in the amount of time specified by its generation period, the packet will be discarded and is considered lost. It has been shown [15] that two percent of the voice packets can be lost and there will be no effect on the voice quality.

Voice is a compressible source, that is, voice can be coded at different rates and the voice quality decreases as the coding rate decreases [16]. There is a technique known as embedded coding [17] which allows the use of variable rate packets to be generated. The idea here is that a packet generated at a high rate can have selected bits stripped away and the resulting packet is of a lower coding rate. The concept of embedded coding or the ability to generate packets at different rates from the same voice source has been assumed in this study.

The basic idea of multirate voice coding for load control is that the voice quality can be traded for network load. When an increase in the traffic intensity on the network has been observed, the voice coding rate will be lowered. By decreasing the voice coding rate the generation period is increased causing fewer voice packets to access the network. The decrease in coding rate reduces the traffic on the network. This system is essentially a feedback loop. The decrease in rate causes less traffic, when there is less traffic the rate is increased, which causes an increase in the traffic.

Figure 4-1 shows an integrated voice and data network which has some method of measuring the traffic and adjusting the voice coding rate. It was found that the number of collisions per millisecond gives a good indication of the amount of traffic on the network, see Section 4.2.

This Chapter is divided into six sections. The first section describes the choice of voice coding rates. The second section describes the feedback algorithm. The third section describes the modifications to the basic CSMA/CD protocol of Chapter 3, Figure 3-1. The fourth section describes how the multirate algorithm was implemented into the existing CSMA/CD simulation model. The fifth section gives the simulation results, and the sixth section describes the system dynamics.

4.1 Choice of Voice Coding Rates

The embedded coder implemented at the Telecommunications and Information Sciences Laboratory (TISL) allows four different voice coding rates, and is based on [17]. The four rates must be chosen such that they are separated by 8 kilo-bits-per-second (kbps). The rates chosen range from 24kbps to 48kbps and the packet length chosen is 768 bits.

The packet length is a constant and there is no specification on which bits are overhead. In this study, the distinction between overhead bits and information bits has not been made. The voice coding rates (kbps) and the corresponding generation periods (milliseconds, ms) are shown below in Table 4-1, recall from Equation (4-1) that the generation period is the packet length divided by the coding rate. Choosing the voice coding rate is equivalent to choosing the generation period, similarly, changing the voice coding rate is equivalent to changing the generation period if the packet length is constant.

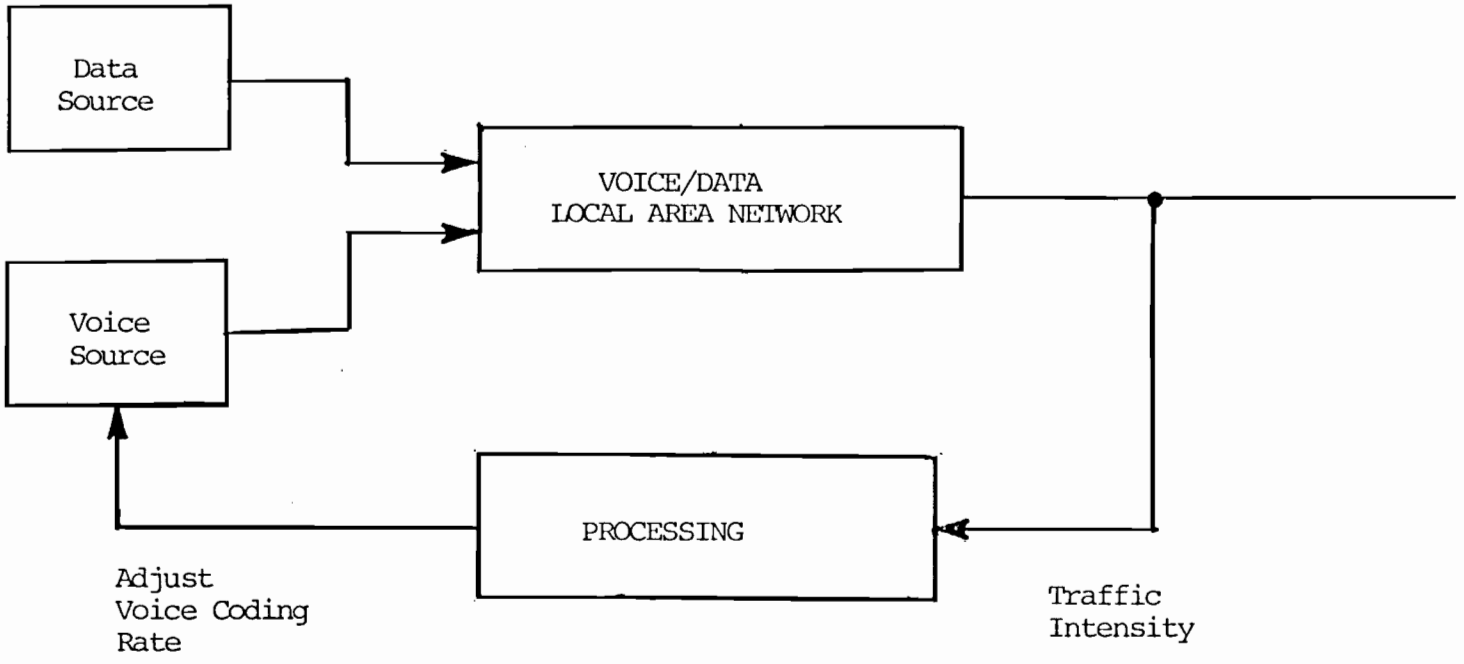


Figure 4-1. Block diagram of a voice/data Local Area Network where the voice coding rate is adjusted according to the traffic intensity on the network.

Voice Coding Rate, kbps	Generation Period, ms
48.0	16.0
40.0	19.2
32.0	24.0
24.0	32.0

Table 4-1 The chosen voice coding rates and the corresponding generation periods, for a voice packet length of 768 bits.

4.2 The Feedback Algorithm

The need for a feedback algorithm was described above. Since changing the voice coding rate changes the amount of traffic on the network, the need for a feedback system is clear. When the traffic increases the voice coding rate is decreased. But decreasing the voice coding rate will decrease the amount of traffic. So, the feedback algorithm must adapt to the changing load conditions. In this section the feedback algorithm will be described. The first subsection will describe the method used to measure the amount of traffic on the network, and the second subsection will describe the feedback equation.

4.2.1 Collisions Per Millisecond as a Load Indicator

An important performance indicator of computer networks is the packet delay, as the traffic on the network increases the packet delay increases. Therefore, the parameter used to measure the trafffic on the network should increase when the packet delay increases. It was found that collisions per millisecond gives a good indication of the packet delay, and therefore gives an indication of the amount of traffic on the network. Figure 4-2 shows that the collisions per millisecond increases at the same rate as the delay.

Collisions per millisecond is easy to measure by counting the jam signals. Recall that when a collision occurs, a jam signal is transmitted to

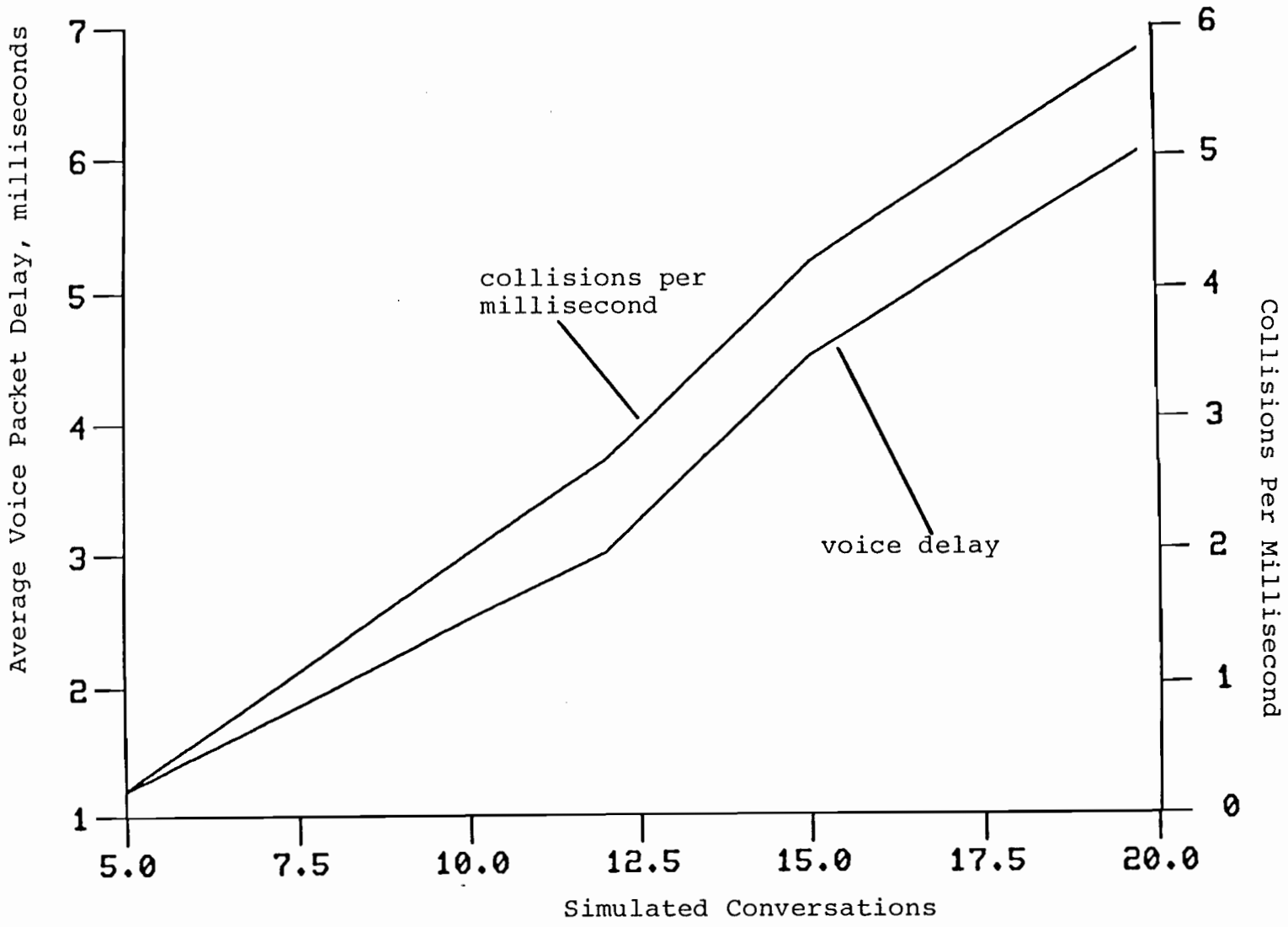


Figure 4-2. Comparison of the voice packet delay in milliseconds to the average rate of collisions per millisecond, with a constant voice coding rate of 48 Kbps, and a constant data load of 15%.

inform all the nodes that a collision has occured. When a jam signal is received, the node will increment a counter, this jam counter will be used to obtain a value for the rate of collisions per millisecond.

4.2.2 The Feedback Equation

The feedback equation must be able to change the voice coding rate dynamically. When the number of collisions per millisecond goes up the voice coding rate must come down, and when the collisions per millisecond goes down the rate must go up. It was found that Equation (4-2) gave the desired characteristics.

$$rate = ravg + q\ (Q - colpms) \qquad (4\text{-}2)$$

where,

rate = the new voice coding rate

ravg = 33000 = the average voice coding rate

q = 13000 = multiplier

Q = 3.3 = average rate of collisions per millisecond

colpms = the measured rate of collisions per millisecond

The values for the three parameters, ravg, q, and Q were chosen by observing the non-multirate network and by experience. The value for the multiplier, q, was chosen to insure that a small change in the (Q-colpms) term will cause a large change in the rate.

The (Q-colpms) term of Equation (4-2) is an error signal. When the colpms is greater than the average value, Q, the term is negative and the rate is driven down. When the error signal is positive, colpms is less than the average, the rate is driven higher. When the error signal is zero, colpms is equal to the average, the rate obtained is the average voice coding rate.

The voice coding rate obtained from Equation (4-2) must be truncated to one of the four possible values given in Table 4-1. The cutoff points used for truncating the value for the voice coding rate obtained from the feedback equation are given below in Table 4-2.

Rate Obtained From the Feedback Equation	Rate is Truncated to
$44 \leq$ rate	48.0
$36 \leq$ rate < 44	40.0
$28 <$ rate < 36	32.0
rate ≤ 28	24.0

Table 4.2 Truncation of the rate obtained from the feedback equation.

4.3 Protocol Modifications for Multirate Voice and Data

The CSMA/CD protocol described in Chapter 3, Figure 3-1, must be modified for voice packets and multirate techniques. The voice packets can be discarded for two reasons. If a voice packet experiences excess collisions or the packet lifetime is exceeded the packet will be discarded. The data packets are discarded only if too many collisions occur. When a voice or data packet experiences too many collisions, greater than 16, the packet is said to be discarded. If a voice packet is not successfully transmitted in the amount of time specified by the generation period then the packet lifetime has been exceeded, and the packet is said to be lost.

To modify the protocol for multirate voice techniques requires that the generation period be set when the voice packet enters the network. By setting the generation period the voice coding rate is effectively being set. In this study the voice packet length is a constant 768 bits, therefore setting the generation period is equivalent to setting the voice coding rate. The modifications to the protocol are shown in the flow diagram of Figure 4-3.

4.4 Implementation Into the Existing CSMA/CD Simulation Model

To implement the multirate voice algorithm into the existing simulation model requires that the Network Model and the Discrete Event Model be modified. In the Network Model the simulation of voice nodes must be done, and in the Discrete Event Model the modifications to the protocol must be incorporated. This section is divided into five subsections. The first describes the new variables and the new files used. The second describes the modifications to the Network Model, the third describes the modifications to the Discrete Event Model. The fourth subsection describes the configuration of the Local Area Network, and the fifth gives some sample outputs of the simulation.

4.4.1 New Variables and File Assignments

In addition to the variables described in Chapter 3, Section 3.2.1, there are several new variables which are outlined below. The new variables include SLAM variables, model variables, and measurement variables.

4.4.1.1 SLAM Variables

The SLAM variables include the attributes and the global variables. The attribute descriptions given in Chapter 3 apply to the attributes here. There are three additional attributes and they have been listed below.

atrib(15) = set to 1 if the voice packet lifetime is exceeded

atrib(16) = set to 1 if packet originated at a data node, set to 2 if packet originated at a voice node

atrib(18) = the generation period for the particular voice packet

In addition to the new attributes, there are several new global variables. The new global variables are listed below.

xx(1) = the next generation period to be used by voice node 1

xx(2) = the next generation period to be used by voice node 2

xx(3) = not used (node 3 is a data node)

xx(4) = the next generation period to be used by voice node 4

xx(5) = not used (node 5 is a data node)

xx(6) = the next generation period to be used by voice node 6

xx(7) = the next generation period to be used by voice node 7

xx(8) = the next generation period to be used by voice node 8

xx(9) = not used (node 9 is a data node)

xx(10) = the next generation period to be used by voice node 10

xx(11) = the next generation period to be used by voice node 11

xx(12) = the next generation period to be used by voice node 12

xx(13) = the next generation period to be used by voice node 13

xx(14) = not used (node 14 is a data node)

xx(15) = the next generation period to be used by voice node 15

xx(16) = the next generation period to be used by voice node 16

xx(17) = not used (node 17 is a data node)

xx(18) = the next generation period to be used by voice node 18

xx(19) = the next generation period to be used by voice node 19

xx(20) = the next generation period to be used by voice node 20

xx(21) - xx(24) not used

xx(25) = the simulated time (ttfin)

xx(26) - xx(29) = not used

xx(30) = the mean interarrival rate for the data nodes

4.4.1.2 Model Variables

The new model variables will be described here. There were several new variables created so that the data load can be changed dynamically during the simulation. These variables are listed below.

loadtim = time between load changes

loadu(numchg) = load used during a specific time range

numlds = number of non-random loads

numrlds = number of random loads

load(numlds) = non-random load array, contains the loads

rload(numrlds) = random load array, contains the possible loads

allrand = if 'yes' then the loads will be determined randomly

numchg = number of load changes or time ranges

There are new variables which are used to calculate the collisions per millisecond and they are listed below. The collisions per millisecond is calculated over a period of time and there is a counter which keeps track of the number of collisions. Every period microseconds a new value is calculated for colpms, the voice coding rate is determined and the colcnt counter is set to zero every period microseconds. The collisions per millisecond and voice coding rate variables are listed below.

period = 32 = amount of time over which the collisions per millisecond is being calculated, in milliseconds

colcnt = counter that keeps track of the number of collisions that occur during a period

colpms = colcnt/period = amount of collisions per millisecond for a specific period

rate = truncated voice coding rate in the DETGEN subroutine

ratenow = calculated rate for a specific period

rateout = truncated voice coding rate in the DETRATE event

choice(4) = the four possible voice coding rates

In addition to the variables described above, there are some miscellaneous variables. These include the line capacity, and are listed below.

capcty = 1.0 = capacity of the line being simulated in mega-bits-per-second

out1 = specifies the file for keeping the time versus voice coding rate for all the nodes, file TR.DAT

out2 = specifies the file for keeping the time versus collisions per millisecond for all the nodes, file CT.DAT

nd8 = specifies the file for keeping the time versus voice coding rate for node 8, file TR8.DAT

nd8x2 = specifies the file for keeping the time versus voice coding rate for node 8 (for plotting purposes), file TR8PLOT.DAT

4.4.1.3 Measurement Variables

The new measurement variables have been listed below.

sumrate = the sum of the voice coding rates used by all the nodes during a specific time range

numrate = the number of the voice coding rates used by all the nodes during a specific time range

sum8 = the sum of the voice coding rates used by node 8 during a specific time range

num8 = the number of the voice coding rates used by node 8 during a specific time range

sumoa = sum of all the voice coding rates, used to compute the overall average rate

numoa = number of all the voice coding rates, used to compute the overall average rates

sum8oa = sum of node 8's rates, used to compute the overall aver age rate for node 8

num8oa = number of node 8's rates, used to compute the overall average rate for node 8

numc = sum of the collisions per millisecond that were calcula- ted during a specific time range

numc = number of the collisions per millisecond that were calcu- lated during a specific time range

sumdel = sum of data delay experienced during a specific time range

numdel = number of data delay experienced during a specific time range

avgrate(numchg) = each element is the average voice coding rate for all the nodes combined during a specific time range

avg8(numchg) = each element is the average voice coding rate for node 8 during a specific time range

avgcpm(numchg) = each element is the average collisions per millisecond during a specific time range

datadel(numchg) = each element is the average data delay during a specific time range

rwdisc(maxsta) = each element contains the number of packets lost in a row due to excessive collisions for a specific node

rwdiscp(maxsta) = each element contains the number of packets lost in a row due to excessive collisions for a specific node

lstpak(maxsta) = each element contains the number of packets lost due to

excess of lifetime for a specific node

rwlst(maxsta) = each element contains the number of packets lost in a row due to excess of lifetime for a specific node

rwlstp(maxsta) = each element contains the number of packets lost in a row due to excess of lifetime for a specific node

4.4.1.4 File Assignments

The file assignments are given below.

Files 1-20	Reserved for the AWAIT Blocks at each station
File 21	Channel QUEUE Block
File 22	Defer file
File 23	Event Calendar

4.4.2 Modifications to the Network Model

The changes that were made to the Network Model of the CSMA/CD simulation will be desribed here. The major change is that voice nodes are being simulated. The voice nodes will generate packets at a periodic rate instead of Poisson. Also, the voice nodes have an additional attribute associated with them, atrib(18), which specifies the generation period. A shortened version of the Network Model that was used to simulate the multirate voice system is given in Figure 4-4, the complete listing is in Appendix B.1.

In this section Figure 4-4 will be discussed. Since most of the code in Figure 4-4 was discussed in Chapter 3 this section will be brief. The new code is the additional statistics being collected, lines 14 through 17, and the voice nodes. The voice nodes are shown on lines 30 through 37, 40 through 47, and 65 through 72. The data node, shown on lines 52 through 58, is the same as the data nodes of the Network Model of Chapter 3 with some excep-

```
GEN,ED FRIEDMAN,MULTIRATE LOAD CNTRL,8/20/84,1,NO,NO;
LIMITS,22,18,750;
;
STAT,1,NODE  1 SYS TIME
        .
        .
        .
STAT,20,NODE 20 SYS TIME
;;
STAT,21,QUEUE DLY   DATA
STAT,22,ACCSS DLY   DATA
STAT,23,CHNL DELAY  DATA
STAT,24,TOTSYS DLY  DATA
STAT,25,QUEUE DLY   VOICE
STAT,26,ACCSS DLY   VOICE
STAT,27,CHNL DLY    VOICE
STAT,28,TOTSYS DLY VOICE
;
NETWORK;
;
      RESOURCE/PACK1,1/PACK2,2/PACK3,3/PACK4,4/PACK5,5/PACK6,6/
               PACK7,7/PACK8,8/PACK9,9/PACK10,10/PACK11,11/
               PACK12,12/PACK13,13/PACK14,14/PACK15,15/PACK16,16/
               PACK17,17/PACK18,18/PACK19,19/PACK20,20;
;
;     MODEL OF THE NUMBER OF STATIONS ON THE ETHER
;     ********************************************
;-------------------------------------------------------------------------------
;<<<<<<<<<<*voice*>>>>>>>>>>
      CREATE,XX(1),2900,1;          create packets at node 1
      ASSIGN,ATRIB(2)=1,
             ATRIB(9)=768.0,
             ATRIB(11)=768.0,
             ATRIB(16)=2,
             ATRIB(18)=XX(1),1;     set attributes
      AWAIT(1),PACK1;               packets wait for previous packet to finish
      ACT,,,CHNL;                   immediate branch to the channel
;-------------------------------------------------------------------------------
;<<<<<<<<<<*voice*>>>>>>>>>>
      CREATE,XX(2),7300,1;
      ASSIGN,ATRIB(2)=2.0,
             ATRIB(9)=768.0,
             ATRIB(11)=768.0,
             ATRIB(16)=2,
             ATRIB(18)=XX(2),1;
      AWAIT(2),PACK2;
      ACT,,,CHNL;
;-------------------------------------------------------------------------------
;-------------------------------------------------------------------------------
;<<<<<<<<<<*data*>>>>>>>>>>
      CREATE,EXPON(XX(30)),2000,1;
      ASSIGN,ATRIB(2)=3.0,
             ATRIB(9)=2048,
             ATRIB(11)=2048,
             ATRIB(16)=1.1;
      AWAIT(3),PACK3;
      ACT,,,CHNL;
;-------------------------------------------------------------------------------
                .
                .
                .
;-------------------------------------------------------------------------------
;<<<<<<<<<<*voice*>>>>>>>>>>
      CREATE,XX(20),8600,1;
      ASSIGN,ATRIB(2)=20,
             ATRIB(9)=768,
             ATRIB(11)=768,
             ATRIB(16)=2,
             ATRIB(18)=XX(20),1;
      AWAIT(20),PACK20;
      ACT,,,CHNL;
;-------------------------------------------------------------------------------
;
;     COMBINE STATIONS TO FORM CHANNEL
;     ********************************
;
CHNL  QUEUE(21);                    dump entities into queue
;
      ACT;                          immediate branch to channel model
;
      EVENT,1;                      gateway to discrete event model, maps
;                                   entities into event 1 ( sense )
;
      ASSIGN,XX(25)=60000000;       set xx(25) equal to ttfin
;
      END;
INIT,0,60000000;
FIN;
```

Figure 4-4. Listing of the Network Model used for the multirate voice simulations on the CSMA/CD Local-Area-Network.

tions. The 16th attribute is set to one, the 16th attribute is used to indicate whether the packet originated at a data node, atrib(16) is set to 1, or at a voice node, atrib(16) is set to 2. In addition, the data node now has a variable, xx(30), for the mean interarrival time, this is done so that the simulated load can be changed dynamically throughout the simulation.

The voice nodes have a constant interarrival time, which is specified as a global variable. The interarrival time for node 1 is xx(1), and that for node 2 is xx(2), and so on. The interarrival time is the generation period for the particular node. This global variable will be changed in the Discrete Event Model whenever a voice coding rate change is required. The voice packets have a smaller packet size and the 16th attribute is set to two. In addition, the voice nodes have an 18th attribute, this attribute is the generation period of the particular packet. The rest of the code in Figure 4-4 was explained in Chapter 3.

4.4.3 Modifications to the Discrete Event Model

The modifications to the protocol given in Figure 4-3 are to be implemented in the Discrete Event Model. Most of the events and subroutines discussed in Chapter 3 remain unchanged, however, the SENSE event, COLLISION subroutine, CALC_WAIT_BACKOFF subroutine, and the FREERSC subroutine have been modified. In addition, there are two new events and one new subroutine. The first new event is the DETRATE event, which calculates a new voice coding rate using the feedback equation every period microseconds (recall that period is 32000 microseconds). The second new event is the LOADCHG event, this event is used to dynamically change the data load and is executed every loadtim microseconds. The new subroutine is the DETGEN subroutine, this subroutine is called for each voice packet when it attempts to access the network for the first time. The DETGEN subroutine determines the generation period.

In this section the modified event and subroutines will be described first, and then the new events and subroutine will be described. The code will not be stepped through due to its length. The events and subroutines will be described using flow diagrams. See Appendix B.2 for a complete listing of the modified and new events and subroutines.

The modified SENSE event is shown in Figure 4-5. The ID is set as before and then a check is made to see if the packet is voice and if this is the first time that the packet has tried to access the network. If the condition is true then the DETGEN subroutine is called so that the generation period can be determined, and the arrival time of the next packet is set. Then the first access attribute is incremented, and then the time left after the collision discrimination period is calculated. Next, there is a check to see if the packet is voice and if the time it has been in the system exceeds the generation period. If this condition is true then the packet is lost, and the 15th attribute is set to 1 and the FREERSC subroutine is called. The rest of the SENSE event is the same as the non-multirate voice case.

The CALC_WAIT_BACKOFF subroutine has been modified slightly. The only difference here is that the backoff is calculated separately for voice and data packets. The voice packets have their backoff calculated using the truncated binary exponential scheme, with the truncation at nine retransmissions. The data packets are truncated at ten retransmissions. The modified CALC_WAIT_BACKOFF is shown in Figure 4-6.

The COLLISION and FREERSC subroutines have also been modified. The only modification to COLLISION is that the colcnt counter is incremented. Recall that the colcnt counter is used to keep track of the number of collisions that have occurred during the amount of time specified by period. The only modifications to FREERSC are that additional statistics are collected.

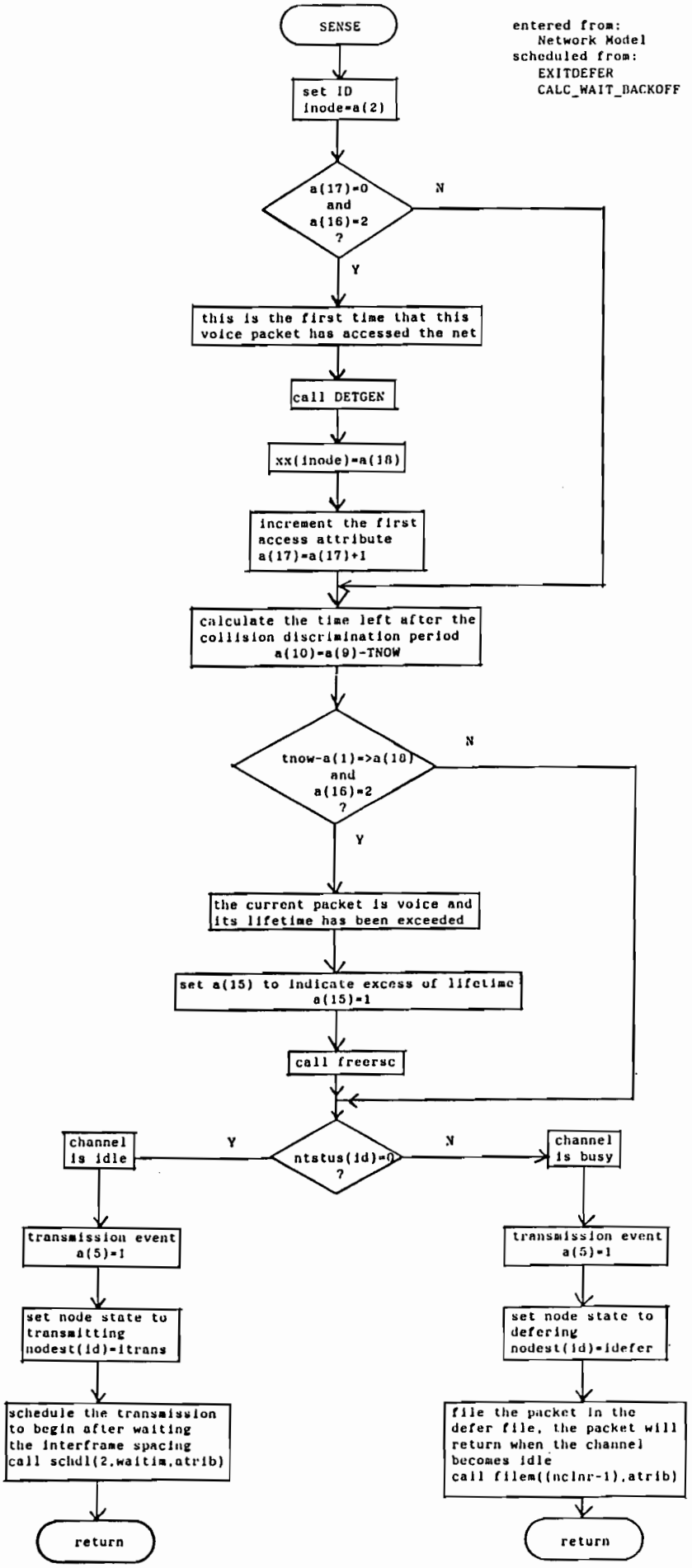


Figure 4-5. Flow diagram of the SENSE event.

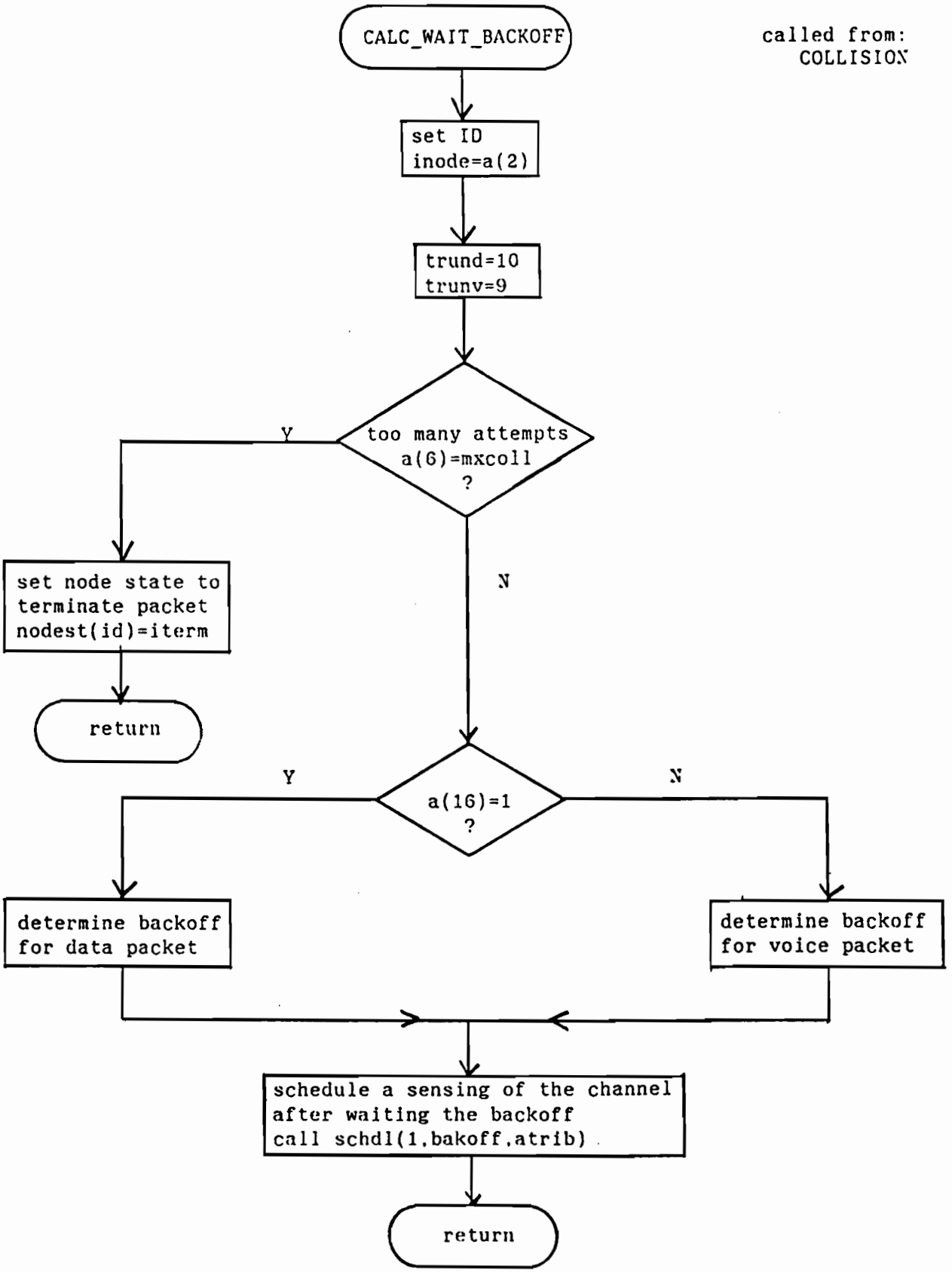


Figure 4-6. Flow diagram of the CALC_WAIT_BACKOFF subroutine.

The DETGEN subroutine is shown in Figure 4-7. This subroutine is called from the SENSE event whenever a voice packet enters SENSE for the first time. This subroutine truncates the rate which was calculated using the feedback equation, ratenow, to one of the four possible rates specified in Table 4-1, see Figure 4-8 for a flow diagram of the truncating algorithm. The truncated rate is then used to calculate the generation period of the packet.

The DETRATE event is shown in Figure 4-9. This event is executed every period microseconds, 32000. The parameters in the feedback equation are set. Then the event is scheduled to be executed again after period microseconds have elapsed. Next the rate of collisions per millisecond is calculated and the colcnt counter is set to zero. Then the voice coding rate that is to be used over the next period microseconds is calculated using the feedback equation and the calculated rate is truncated using the algorithm of Figure 4-8.

The LOADCHG event is shown in the flow diagram of Figure 4-10. This event is executed every loadtim microseconds. When LOADCHG is executed the load being simulated by the data nodes is changed. Therefore, every loadtim microseconds a new load is being simulated. This way the traffic on the network is changing throughout the simulation. The load can be the same throughout the simulation by initializing the rload and load arrays to the same value. To have the load change randomly, the user would choose the sequence of loads desired and place them in the rload array. The allrand variable would be set to 'yes' and the loadtim variable would be set to the amount of time that each particular load is to be simulated.

This completes the description of how the CSMA/CD simulation model of Chapter 3 was modified to simulate the multirate voice and data system. The

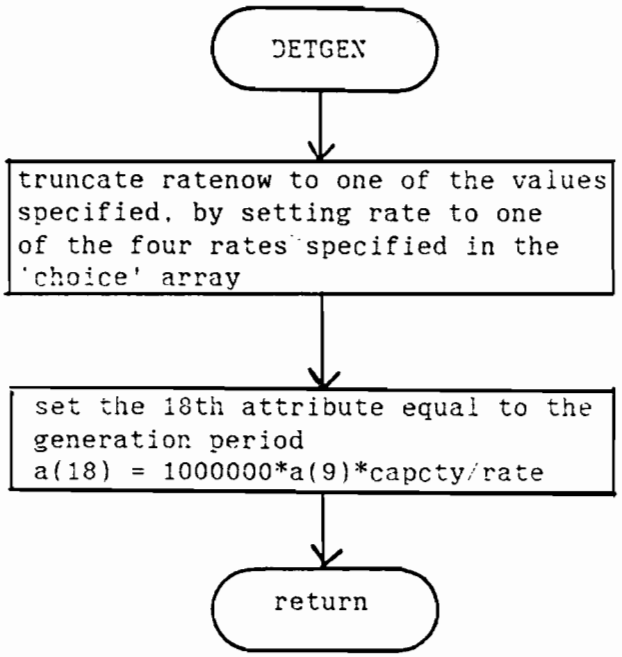


Figure 4-7. Flow diagram of the DETGEN subroutine.

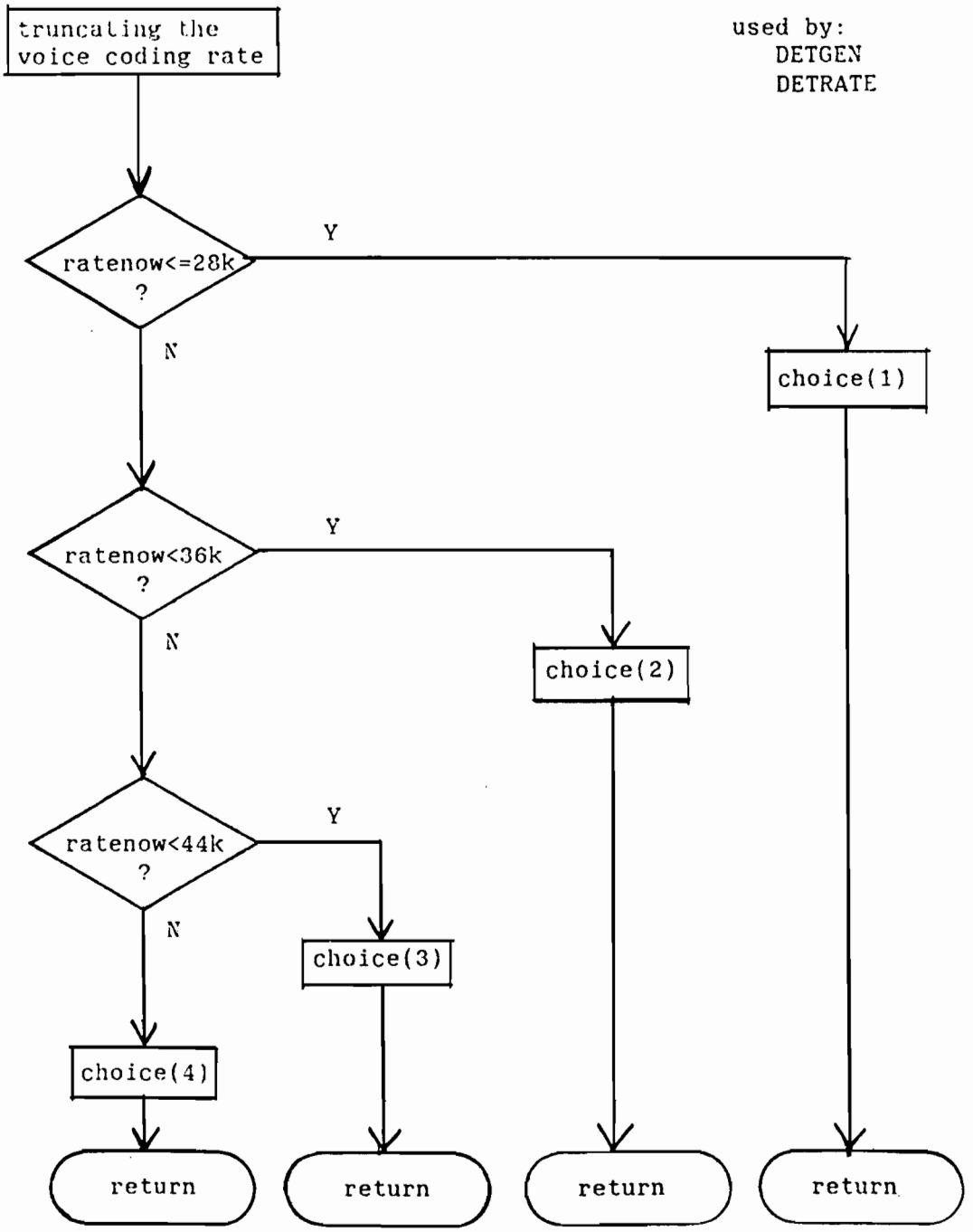


Figure 4-8. Flow diagram of the process required to truncate the calculated value of the voice coding rate to one of the four possible rates.

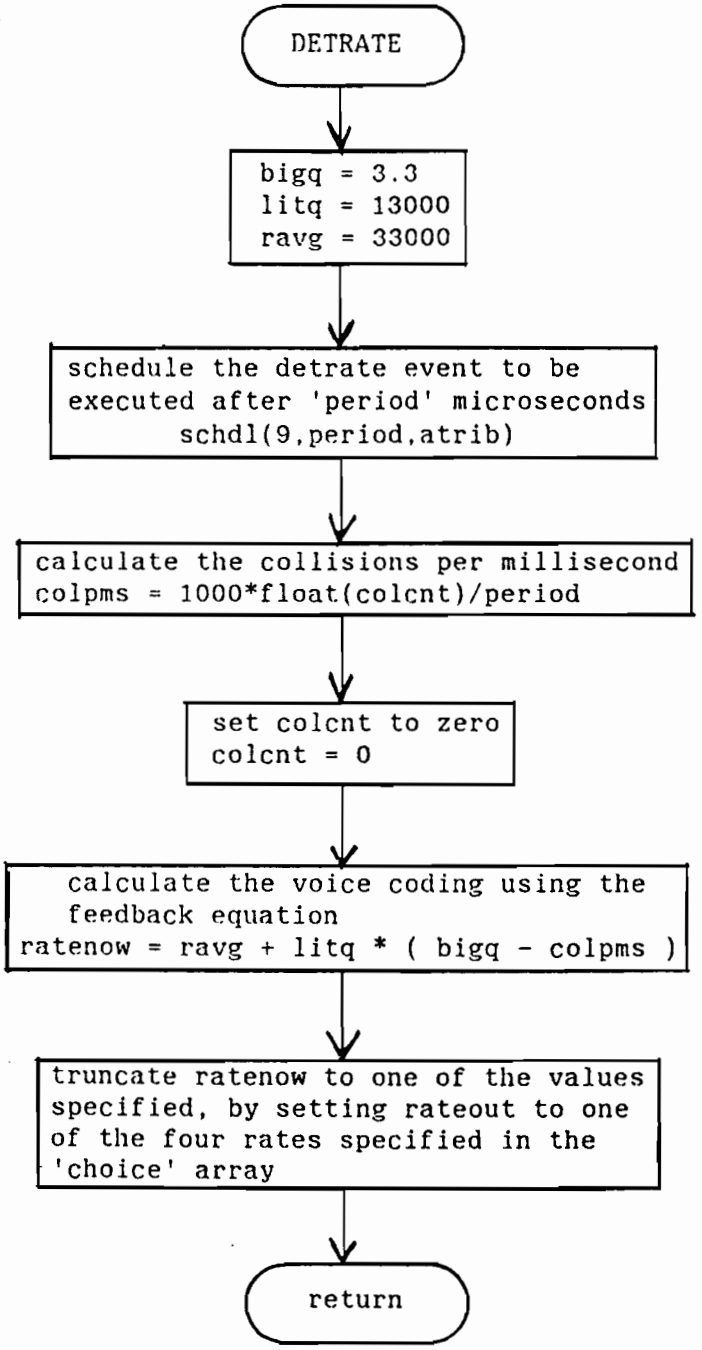


Figure 4-9. Flow diagram of the DETRATE event.

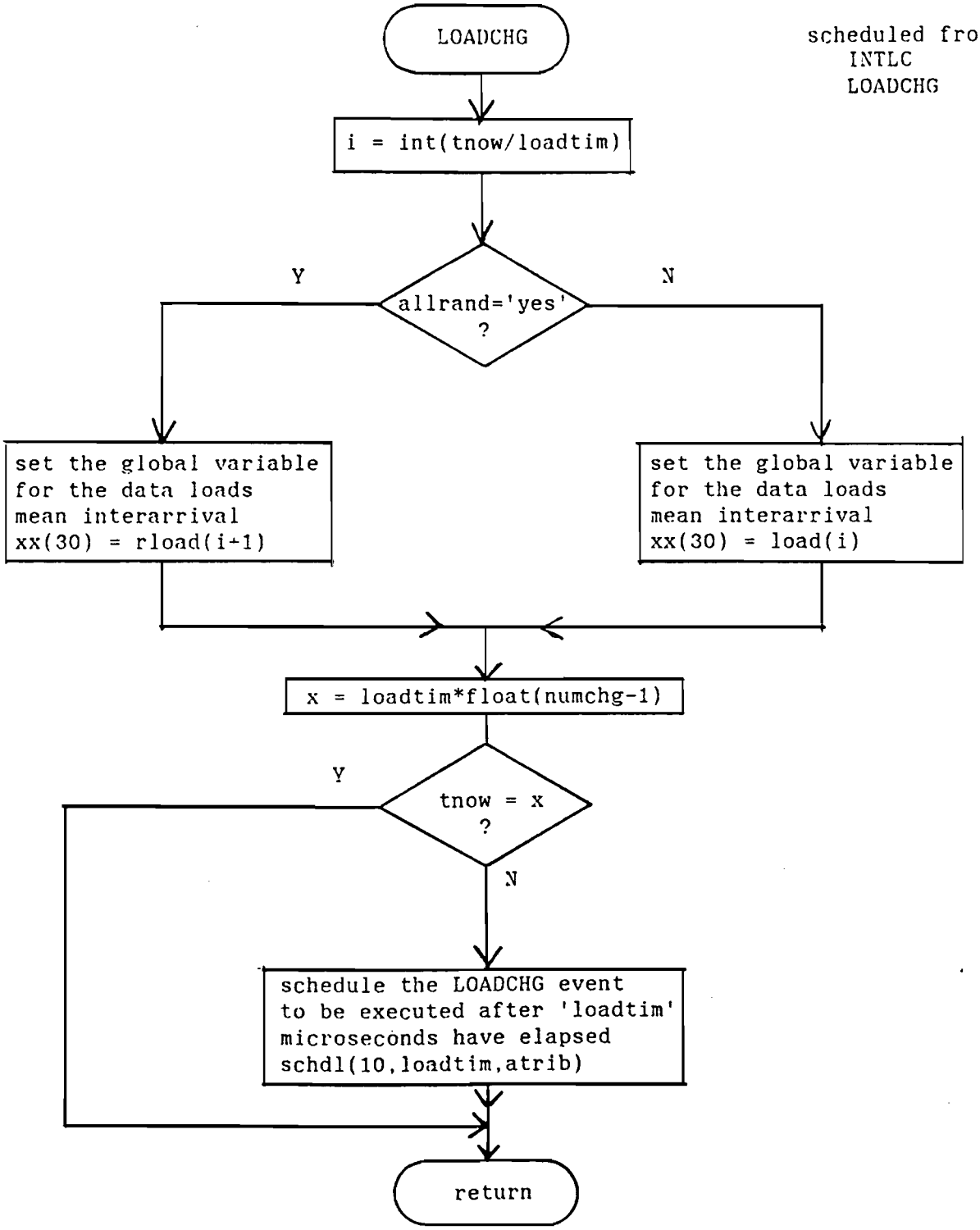


Figure 4-10. Flow diagram of the LOADCHG event.

simulation of voice nodes was described in Section 4.4.2, where a shortened version of the Network Model was stepped through. The portions of the Discrete Event Model that were modified have been described, and the new events and subroutine were discussed. The last two topics of this section give the configuration of the network and some sample outputs, respectively. The configuration to be presented was used to obtain the simulation results of sections 4.5 and 4.6.

4.4.4 System Configuration

The configuration used to obtain the simulation results is given below. This configuration is set up in the INTLC subroutine and the PARAMS file. The INTLC subroutine has been expanded due to the many new variables, see Appendix B.3 for a complete listing of the modified INTLC subroutine. The PARAMS file has also been expanded, see Appendix B.4 for a complete listing.

- line capacity: 1 mega-bit-per-second
- bus length: approximately 1 kilometer (4.5 microseconds)
- slot time: round-trip bus delay = 9.0 microseconds
- backoff algorithm for data: binary exponential backoff truncated at 2^{10}
- backoff algorithm for voice: binary exponential backoff truncated at 2^{9}
- jam time: 4.8 microseconds
- interframe spacing: 9.6 microseconds
- data packet length: 4096 bits
- voice packet length: 768 bits
- five data nodes generating Poisson arrivals

4.4.5 Sample Outputs

In this section there will be two types of output discussed for the constant data load case and the random data load case. The first output is the usual SLAM Summary Report and the second is the output from the OTPUT subroutine, see Appendix B.5 for a listing of the OTPUT subroutine.

The constant data load case will be discussed first. Figures 4-11 and 4-12 show sample outputs of the constant data load case. Figure 4-11 is the SLAM Summary Report and Figure 4-12 is the output from the OTPUT subroutine. The SLAM Summary Report contains the usual information, the system delay, the file statistics, service activity statistics, and resource statistics. The output from the OTPUT subroutine contains all of the information that was discussed in Chapter 3 for the non-multirate case with some additional information. In addition to the information that was presented in Chapter 3, there is more information presented on the number of packets discarded due to excessive collisions. The percent discarded for each node and the number of packets discarded in a row is given. The number of packets, the percentage, and the number of packets lost in row due to excess of lifetime is presented. The combined, number and percentage of packets discarded and lost is presented. A table is presented which shows the average voice coding rate, the data load, the average rate of collisions per millisecond, and the average data delay for specific time ranges. In the example shown here the information in the table is presented every three seconds, so the loadtim variable was set to three seconds. Next, the overall average voice coding rate for all the nodes combined and that for a single node, node 8, is given. The average percent of the load supplied by the data nodes is given. The average collisions per millisecond and the data delay are presented.

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT MULTIRATE LOAD CNTRL BY ED FRIEDMAN

DATE 8/20/1984 RUN NUMBER 1 OF 1

CURRENT TIME 0.1500E+08
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
NODE 1 SYS TIME	0.2494E+04	0.3098E+04	0.1238E+01	0.7775E+03	0.1964E+05	922
NODE 2 SYS TIME	0.1669E+04	0.1776E+04	0.1064E+01	0.7780E+03	0.1862E+05	922
NODE 3 SYS TIME	0.3618E+04	0.3231E+04	0.8930E+00	0.2058E+04	0.2316E+05	239
.						
NODE 17 SYS TIME	0.3395E+04	0.2374E+04	0.6994E+00	0.2058E+04	0.1995E+05	208
QUEUE DLY DATA	0.1795E+03	0.1259E+04	0.7014E+01	0.0000E+00	0.1893E+05	1096
ACCSS DLY DATA	0.1283E+04	0.2823E+04	0.2201E+01	0.9500E+01	0.2946E+05	1096
CHNL DELAY DATA	0.2048E+04	0.0000E+00	0.0000E+00	0.2048E+04	0.2048E+04	1096
TOTSYS DLY DATA	0.3510E+04	0.3147E+04	0.8967E+00	0.2058E+04	0.3151E+05	1096
QUEUE DLY VOICE	0.9497E+01	0.1489E+03	0.1568E+02	0.0000E+00	0.5324E+04	11066
ACCSS DLY VOICE	0.8907E+03	0.2190E+04	0.2458E+01	0.9500E+01	0.1859E+05	11066
CHNL DLY VOICE	0.7680E+03	0.0000E+00	0.0000E+00	0.7680E+03	0.7683E+03	10993
TOTSYS DLY VOICE	0.1685E+04	0.2303E+04	0.1367E+01	0.7775E+03	0.2132E+05	11066

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	AWAIT	0.0015	0.0385	1	0	24.1015
2	AWAIT	0.0003	0.0182	1	0	5.4182
.						
17	AWAIT	0.0017	0.0410	1	0	121.4363
29	QUEUE	0.0000	0.0000	0	0	0.0000
30		0.5427	0.9532	8	0	422.9107
31	CALENDAR	19.7706	0.5513	47	18	290.1824

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
0	CHNL QUEUE	1	0.0000	0.0000	0	0.0000	18002.0000	0.0000	

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTILIZATION	CURRENT UTILIZATION
1	PACK1	1	0.1518	0.3588	1	0
2	PACK2	1	0.1022	0.3030	1	0
.						
17	PACK17	1	0.0454	0.2082	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	PACK1	1	0.8482	0	1
2	PACK2	1	0.8978	0	1
.					
17	PACK17	1	0.9546	0	1

Figure 4-11. Sample output of the SLAM Summary Report for the constant data load case, data load of 15%.

```
time spent sending good packets from station  1 = 696376.00      thput = 0.046
time spent sending good packets from station  2 = 705792.06      thput = 0.047
                                    .
                                    .
time spent sending good packets from station 17 = 425984.00      thput = 0.028

total time spent with good packets = 10687233.00      total throughput = 0.712


total # of collisions that occured at station   1 = 1643
total # of collisions that occured at station   2 = 1285
                                    .
                                    .
total # of collisions that occured at station  17 =  418

total # of collisions = 14431  average collisions per millisecond = 0.9621


# of packets successful on the first attempt to access net from  1 =   9
# of packets successful on the first attempt to access net from  2 =   2
                                    .
                                    .
# of packets successful on the first attempt to access net from 17 =  54

total # of 1st access = 3864       % of 1st access = 31.7711


# of packets successful on the  1st  attempt = 8050     percentage = 66.190
# of packets successful on the  2nd  attempt = 1277     percentage = 10.500
# of packets successful on the  3rd  attempt =  623     percentage =  5.123
# of packets successful on the  4th  attempt =  710     percentage =  5.838
# of packets successful on the  5th  attempt =  372     percentage =  3.059
# of packets successful on the  6th  attempt =  300     percentage =  2.467
# of packets successful on the  7th  attempt =  195     percentage =  1.603
# of packets successful on the  8th  attempt =  149     percentage =  1.225
# of packets successful on the  9th  attempt =  139     percentage =  1.143
# of packets successful on the 10th  attempt =  145     percentage =  1.192
# of packets successful on the 11th  attempt =  106     percentage =  0.872
# of packets successful on the 12th  attempt =   63     percentage =  0.518
# of packets successful on the 13th  attempt =   25     percentage =  0.206
# of packets successful on the 14th  attempt =    6     percentage =  0.049
# of packets successful on the 15th  attempt =    1     percentage =  0.008
# of packets successful on the 16th  attempt =    1     percentage =  0.008

            % successful on 1st attempt =  66.1898


total # of packets transmitted from station  1 = 922
total # of packets transmitted from station  2 = 922
                                    .
                                    .
total # of packets transmitted from station 17 = 208

total # of packets transmitted =         12162


---packets discarded due to excessive collisions---
node  1:  # of packets discarded = 0   percent = 0.00   discarded in a row = 0
node  2:  # of packets discarded = 0   percent = 0.00   discarded in a row = 0
                                    .
                                    .
node 17:  # of packets discarded = 0   percent = 0.00   discarded in a row = 0

total # of discarded packets = 0     % of packets discarded = 0.0000


total # of successful bits transmitted =        10687232

total # of unsuccessful (discarded) bits =    56064


---packets discarded due to excess of lifetime---
node   1:  # of packets lost = 15     percentage =  1.63     lost in a row =  2
node   2:  # of packets lost =  3     percentage =  0.33     lost in a row =  1
                                    .
                                    .
node  17:  # of packets lost =  0     percentage =  0.00     lost in a row =  0


average percentage of lost packets = 0.6002       total # of packets lost = 73


------ total of lost voice packets ------
    number =        73     percentage =  0.6002


average rate                                    average   average
total  node 8  time range    % load   colpms    data delay, ms
 46.9   47.4    0.0-3.0       15.0   1.0386       3.5609
 47.6   47.6    3.0-6.0       15.0   0.9860       3.4825
 47.7   47.7    6.0-9.0       15.0   0.8059       3.3772
 47.5   47.5    9.0-12.0      15.0   0.9153       3.1992
 47.2   47.2   12.0-15.0      15.0   1.0662       3.8762

          overall average rate = 47.3675
overall average rate for node 8 = 47.4620
          average percent load = 15.0000
        overall average colpms =  0.9624
      average data delay in ms =  3.4992
```

Figure 4-12. Sample output of the OTPUT results for the constant data load case, data load of 15%.

For the random data load case the outputs are presented in Figure 4-13 and 4-14. The SLAM Summary Report is the same as the constant data load case, however, the OTPUT results are slightly different. The only difference is in the table presented at the bottom of the output. In the random case there is a different data load for each time range, also the simulation was run longer. In the constant load case the simulation was run for 15 seconds of real time and in the random load case the simulation was run for 60 seconds.

This completes the description of the multirate voice coding system. The choice of coding rates, the feedback algorithm, and the protocol modifications have been presented. The implementation of the multirate system into the existing CSMA/CD simulation model has been discussed. And the form of the output from the simulation has been given. In the following section the simulation results will be presented.

4.5 Simulation Results

The multirate voice coding system was extensively tested using the simulation model described in the previous sections. The results of the simulation experiments are presented here. The results are presented in two sections. The first section gives the results of the constant data load case and the second section gives the results of the random data load case. The constant load case is compared to the case where no multirate coding is used. And the random load case is compared to the constant load case.

There are five data nodes on the network in all the cases. The data nodes are numbered 3, 5, 9, 14 and 17. The combined load simulated by the data nodes is either 15% or the load specified in the random case.

The non-multirate case was run so that a comparison can be made with the

S L A M S U M M A R Y R E P O R T

SIMULATION PROJECT MULTIRATE LOAD CNTRL BY ED FRIEDMAN

DATE 8/20/1984 RUN NUMBER 1 OF 1

CURRENT TIME 0.6000E+08
STATISTICAL ARRAYS CLEARED AT TIME 0.0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

	MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NUMBER OF OBSERVATIONS
NODE 1 SYS TIME	0.3237E+04	0.3530E+04	0.1090E+01	0.7760E+03	0.2871E+05	3076
NODE 2 SYS TIME	0.3759E+04	0.4191E+04	0.1115E+01	0.7760E+03	0.3350E+05	3073
.						
.						
NODE 17 SYS TIME	0.3572E+04	0.3771E+04	0.1056E+01	0.2056E+04	0.4436E+05	869
QUEUE DLY DATA	0.2832E+03	0.1939E+04	0.6846E+01	0.0000E+00	0.4092E+05	4435
ACCSS DLY DATA	0.1395E+04	0.3908E+04	0.2802E+01	0.8000E+01	0.4770E+05	4434
CHNL DELAY DATA	0.2049E+04	0.0000E+00	0.0000E+00	0.2048E+04	0.2050E+04	4428
TOTSYS DLY DATA	0.3724E+04	0.4400E+04	0.1182E+01	0.2056E+04	0.6411E+05	4434
QUEUE DLY VOICE	0.3599E+02	0.4788E+03	0.1331E+02	0.0000E+00	0.1750E+05	36903
ACCSS DLY VOICE	0.2490E+04	0.3797E+04	0.1525E+01	0.8000E+01	0.3174E+05	36902
CHNL DLY VOICE	0.7685E+03	0.0000E+00	0.0000E+00	0.7680E+03	0.7700E+03	36466
TOTSYS DLY VOICE	0.3321E+04	0.3906E+04	0.1176E+01	0.7760E+03	0.3460E+05	36902

FILE STATISTICS

FILE NUMBER	ASSOCIATED NODE TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAITING TIME
1	AWAIT	0.0014	0.0373	1	0	27.1175
2	AWAIT	0.0027	0.0518	1	0	52.5822
.						
.						
17	AWAIT	0.0033	0.0573	1	0	227.6430
21	QUEUE	0.0000	0.0000	0	0	0.0000
22		0.9139	1.7491	14	1	410.7906
23	CALENDAR	20.3125	1.3343	71	19	185.4189

SERVICE ACTIVITY STATISTICS

ACTIVITY INDEX	START NODE LABEL/TYPE	SERVER CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	CURRENT UTILIZATION	AVERAGE BLOCKAGE	MAXIMUM IDLE TIME/SERVERS	MAXIMUM BUSY TIME/SERVERS	ENTITY COUNT
0	CHNL QUEUE	1	0.0000	0.0000	0	0.0000	27800.0000	0.0000	

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTILIZATION	CURRENT UTILIZATION
1	PACK1	1	0.1646	0.3708	1	0
2	PACK2	1	0.1898	0.3922	1	0
.						
.						
17	PACK17	1	0.0484	0.2147	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	PACK1	1	0.8354	0	1
2	PACK2	1	0.8102	0	1
.					
.					
17	PACK17	1	0.9516	0	1

Figure 4-13. Sample output of the SLAM Summary Report for the random data load case, average data load of 15%.

```
time spent sending good packets from station  1 = 2329484.00      thput = 0.039
time spent sending good packets from station  2 = 2321782.25      thput = 0.039

time spent sending good packets from station 17 = 1778120.00      thput = 0.030

total time spent with good packets =  37098252.00       total thput = 0.618

total # of collisions that occured at station  1 = 11033
total # of collisions that occured at station  2 = 11072

total # of collisions that occured at station 17 =  1483

total # of collisions = 133693  average collisions per millisecond = 2.2282

# of packets successful on the first attempt to access net from  1 =  104
# of packets successful on the first attempt to access net from  2 =  281

# of packets successful on the first attempt to access net from 17 =  293

        total # of 1st access =      9134       % of 1st access =  22.0970

# of packets successful on the  1st  attempt =    15581     percentage = 37.694
# of packets successful on the  2nd  attempt =     3596     percentage =  8.699
# of packets successful on the  3rd  attempt =     3019     percentage =  7.304
# of packets successful on the  4th  attempt =     3702     percentage =  8.956
# of packets successful on the  5th  attempt =     2353     percentage =  5.692
# of packets successful on the  6th  attempt =     1795     percentage =  4.342
# of packets successful on the  7th  attempt =     1529     percentage =  3.699
# of packets successful on the  8th  attempt =     2024     percentage =  4.896
# of packets successful on the  9th  attempt =     2538     percentage =  6.140
# of packets successful on the 10th  attempt =     2858     percentage =  6.914
# of packets successful on the 11th  attempt =     1350     percentage =  3.266
# of packets successful on the 12th  attempt =      550     percentage =  1.331
# of packets successful on the 13th  attempt =      255     percentage =  0.617
# of packets successful on the 14th  attempt =      102     percentage =  0.247
# of packets successful on the 15th  attempt =       37     percentage =  0.090
# of packets successful on the 16th  attempt =       20     percentage =  0.048

            % successful on 1st attempt =  37.6935

total # of packets transmitted from station  1 =  3076
total # of packets transmitted from station  2 =  3073

total # of packets transmitted from station 17 =   869

                total # of packets transmitted = 41336

---packets discarded due to excessive collisions---
node  1:  # of packets discarded = 2   percent = 0.07   discarded in a row = 1
node  2:  # of packets discarded = 3   percent = 0.10   discarded in a row = 1

node 17:  # of packets discarded = 1   percent = 0.12   discarded in a row = 1

    total # of discarded packets =     27     % of packets discarded =   0.0653

total # of successful bits transmitted =     37076012

total # of unsuccessful (discarded) bits =   345600

---packets discarded due to excess of lifetime---
node   1:  # of packets lost = 43     percent = 1.40      lost in a row =  2
node   2:  # of packets lost = 49     percent = 1.59      lost in a row =  2

node  17:  # of packets lost =  0     percent = 0.00      lost in a row =  0

average percentage of lost packets = 0.9991    total # of packets lost = 413

----- total of lost voice packets -----
    number =       434     percentage =  1.0499

average rate                                          average   average
total  node 8  time range     % load     colpms    data delay, ms
 47.5   47.5    0.0-5.0        15.0     0.9503       3.5425
 48.0   48.0    5.0-10.0        5.0     0.2388       2.6706
 46.0   46.0   10.0-15.0       20.0     1.3650       4.3564
 43.2   43.3   15.0-20.0       15.0     2.1538       4.3103
 42.6   43.1   20.0-25.0        5.0     2.3260       3.1127
 41.1   41.3   25.0-30.0       10.0     2.6380       2.9815
 36.3   36.3   30.0-35.0       25.0     3.2163       3.2988
 38.2   37.9   35.0-40.0        5.0     2.7708       2.9585
 37.8   37.3   40.0-45.0       15.0     2.9473       3.0895
 37.8   37.4   45.0-50.0       20.0     2.9012       3.7115
 39.7   40.0   50.0-55.0       25.0     2.6492       4.3714
 40.4   40.5   55.0-60.0       20.0     2.5742       4.0168

          overall average rate = 41.5659
overall average rate for node 8 = 42.0598
          average percent load = 15.0000
        overall average colpms =  2.2276
      average data delay in ms =  3.5350
```

Figure 4-14. Sample output of the OTPUT results for the random data load case, average data load of 15%.

multirate case. In the non-multirate case the voice coding rate is 48 kbps for all the voice nodes, and the data load is 15%.

4.5.1 Constant Data Load

In this section, the results of the constant data load simulation runs will be explained. Each result is presented as a figure showing the performance indicator versus the number of simulated conversations. Each voice node on the network represents a conversation.

The percentage of lost voice packets is an important performance indicator for integrated voice/data networks. It gives an indication of the feasibility in using packet voice on a particular network. It has been shown that the voice quality degrades rapidly when the percentage of lost voice packets exceeds the 2% level [15]. In Figure 4-15 the percentage of lost voice packets versus the number of simulated conversations is shown for the multirate coding case and the non-multirate coding case. When the traffic on the network increases, the voice coding rate will be decreased. A decrease in the voice coding will cause less traffic on the network, and therefore reduces the percentage of lost voice packets at high loads.

The result shows a substantial increase in the number of simulated conversations over the case where no multirate voice coding is used. The 2% level of lost voice packets when no multirate coding is used is at approximately 12 conversations, and that with multirate coding is approximately 22 conversations. A comparison of the constant coding rate case to the results given in [12, 13, 14] shows good agreement.

The curve of the lost packet percentage for the multirate case is not smooth due to the coding rate changes. When the number of simulated conversations increases, the percentage of lost voice packets is expected to in-

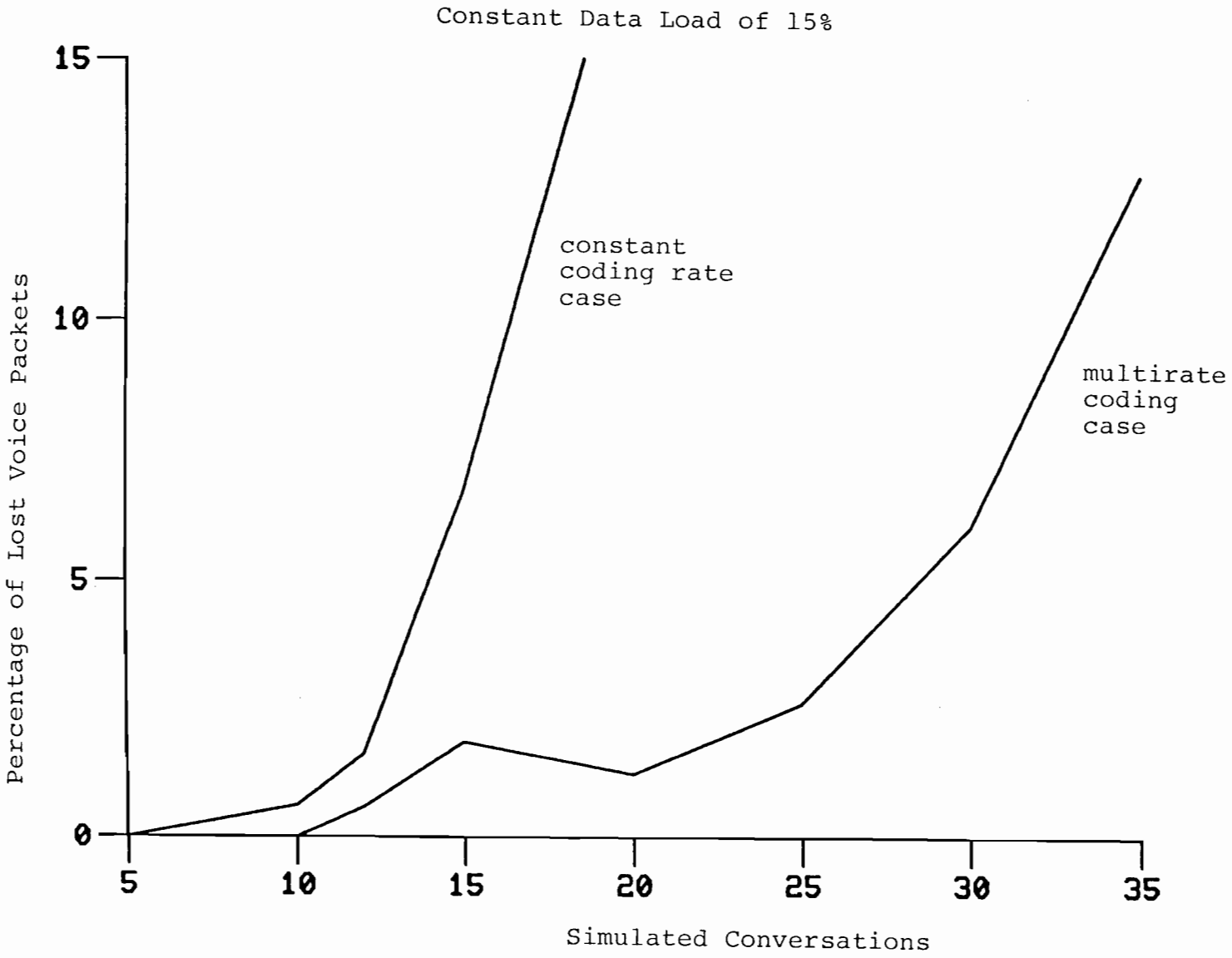


Figure 4-15. Comparison of the percentage of lost voice packets for the multirate and the non-multirate cases.

crease. However, the percentage of lost packets drops from 1.9 when 15 conversations are simulated to 1.3 when 20 conversations are simulated. This occurs because the coding rate decreases when the number of conversations increases, which causes fewer voice packets to access the network. Thereby, decreasing the percentage of lost packets, allowing a greater number of conversations to take place.

The voice delay and the collisions per millisecond are shown in Figure 4-16. The figure shows that the collisions per millisecond increases with the voice packet delay, which verifies that the collisions per millisecond is giving a good indication of the network traffic. The voice delay decreases from 9.5 milliseconds when 25 conversations are simulated, to 8.5 milliseconds when 30 conversations are simulated. The delay should increase as the number of conversations is increased, however, this figure does not show how many packets were lost. When a packet is lost the delay statistics are not collected for that particular packet. So, at the higher loads more packets are lost and the packets that are discarded are not included in the delay statistics. The voice packet delay is again shown in Figure 4-17, where a comparison is made to the non-multirate case. From Figure 4-17 it appears that the voice delay is greater for the multirate case, however, this graph is deceiving since the delay statistics are not collected for lost voice packets. The voice delay curves show the expected delay for voice packets that have been successfully transmitted over the network.

The throughput is another important network performance indicator. The throughput gives an indication of how much of the network capacity is being utilized. The throughput is calculated as the time the network was carrying packets successfully divided by the total time that was simulated. A comparison of the throughput for the non-multirate case and the multirate case is

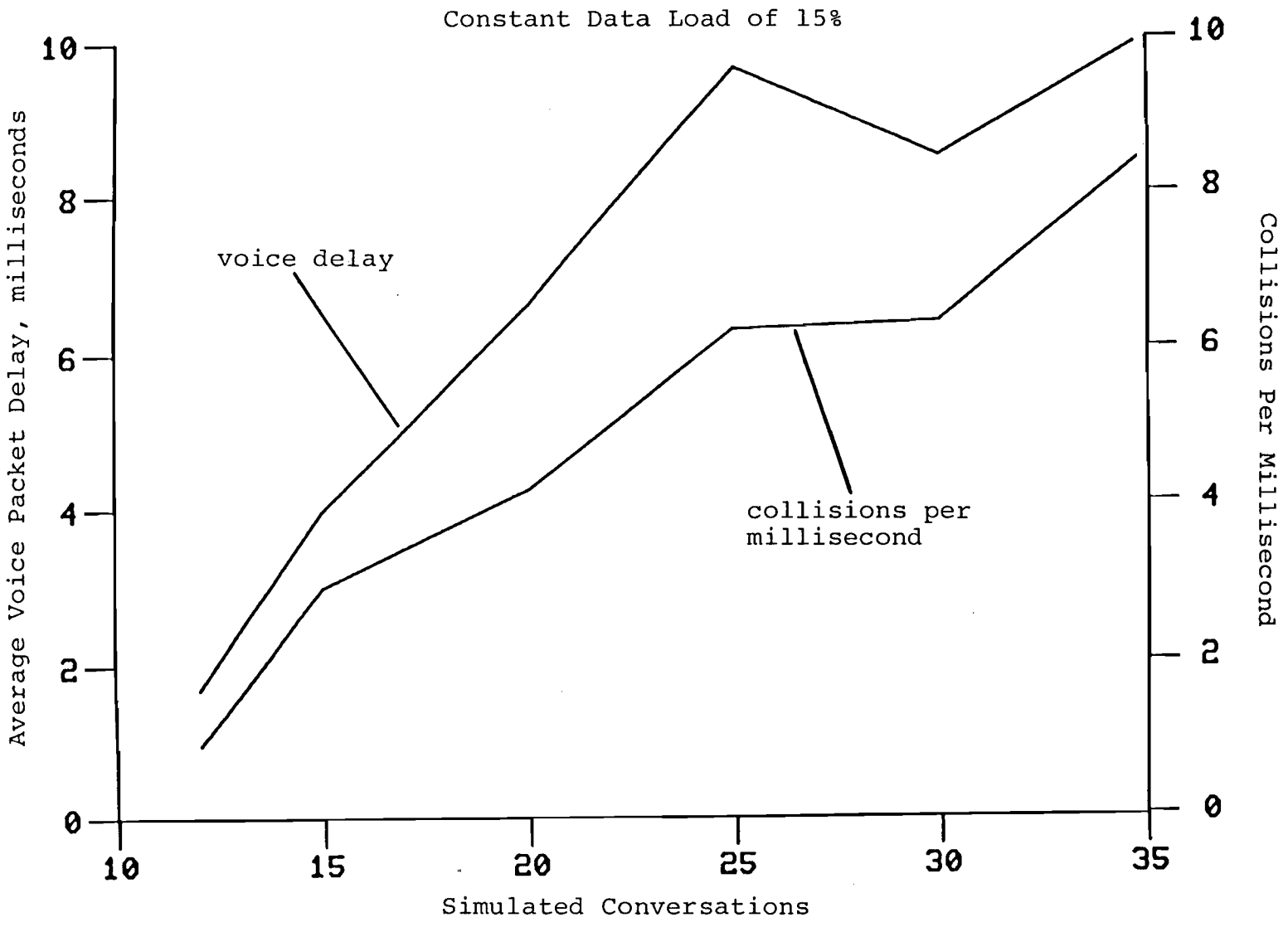


Figure 4-16. Rate of collisions per millisecond and the average voice packet system delay versus the number of simulated conversations.

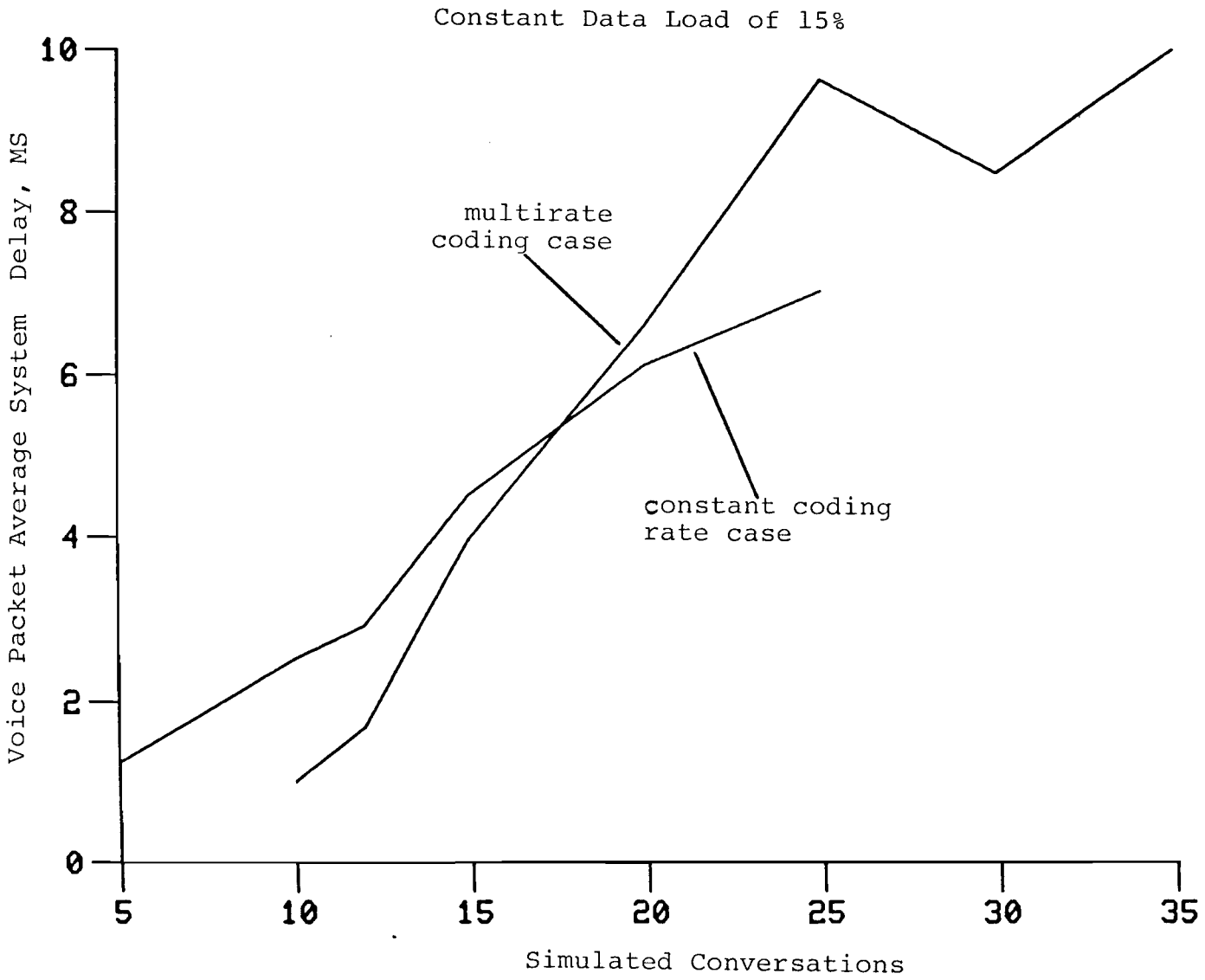


Figure 4-17. Comparison of the voice packet delay for the multirate and non-multirate cases.

shown in Figure 4-18. When the load is high, the multirate case gives a lower total throughput than the non-multirate case. This is because when the load is high the voice coding rate is decreased which causes less voice packets to attempt transmission.

The segmental signal-to-noise ratio gives an indication of the voice quality [18]. The segmental signal-to-noise for the non-multirate case is 30 dB. The voice quality, and therefore the segmental signal-to-noise should drop as the voice coding rate decreases. The voice coding rate should drop down to the lowest rate as the number of conversations increases. The voice coding rate and segmental signal-to-noise are shown in Figure 4-19, where the expected result was obtained.

The data packet delay is another performance indicator. The data delay should remain as low as possible from the point of view of a network user [19]. A comparison of the data packet delay for the constant coding rate case and the multirate coding case is shown in Figure 4-20. As seen in Figure 4-20, the data packet delay is decreased when the multirate voice coding techniques are employed.

4.5.2 Random Data Load Case

In this section the results of simulating data loads that vary over time will be presented. For this experiment the average data load was 15%. The load was changed every five seconds. The sequence of data loads is shown in the table at the bottom of Figure 4-14. The results for this case should be approximately the same as the constant load case. In the figures the results for the constant load case and the random load case are presented so that a comparison can be made. The results are given in Figures 4-21 through 4-27.

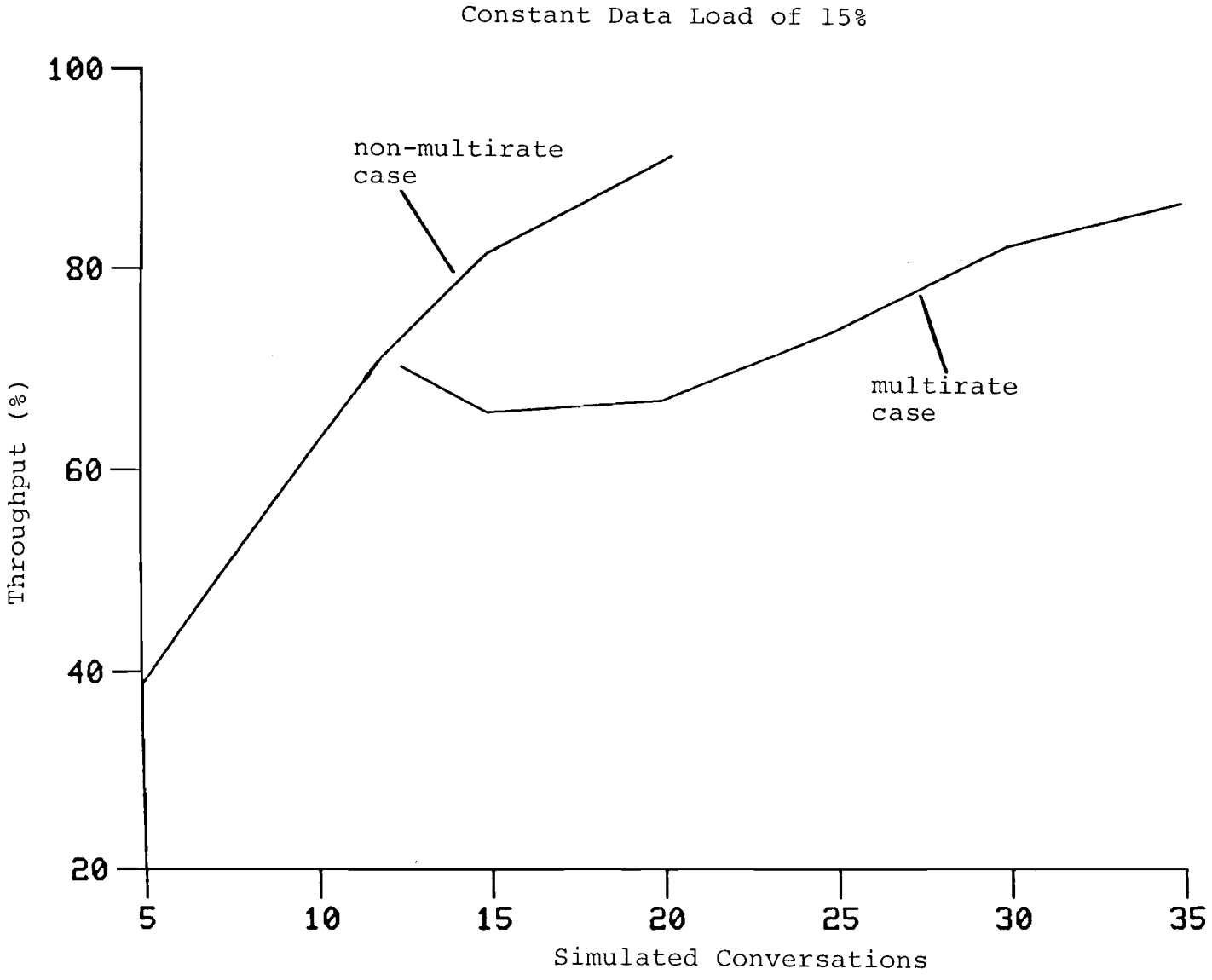


Figure 4-18. Comparison of throughput for the multirate and non-multirate cases.

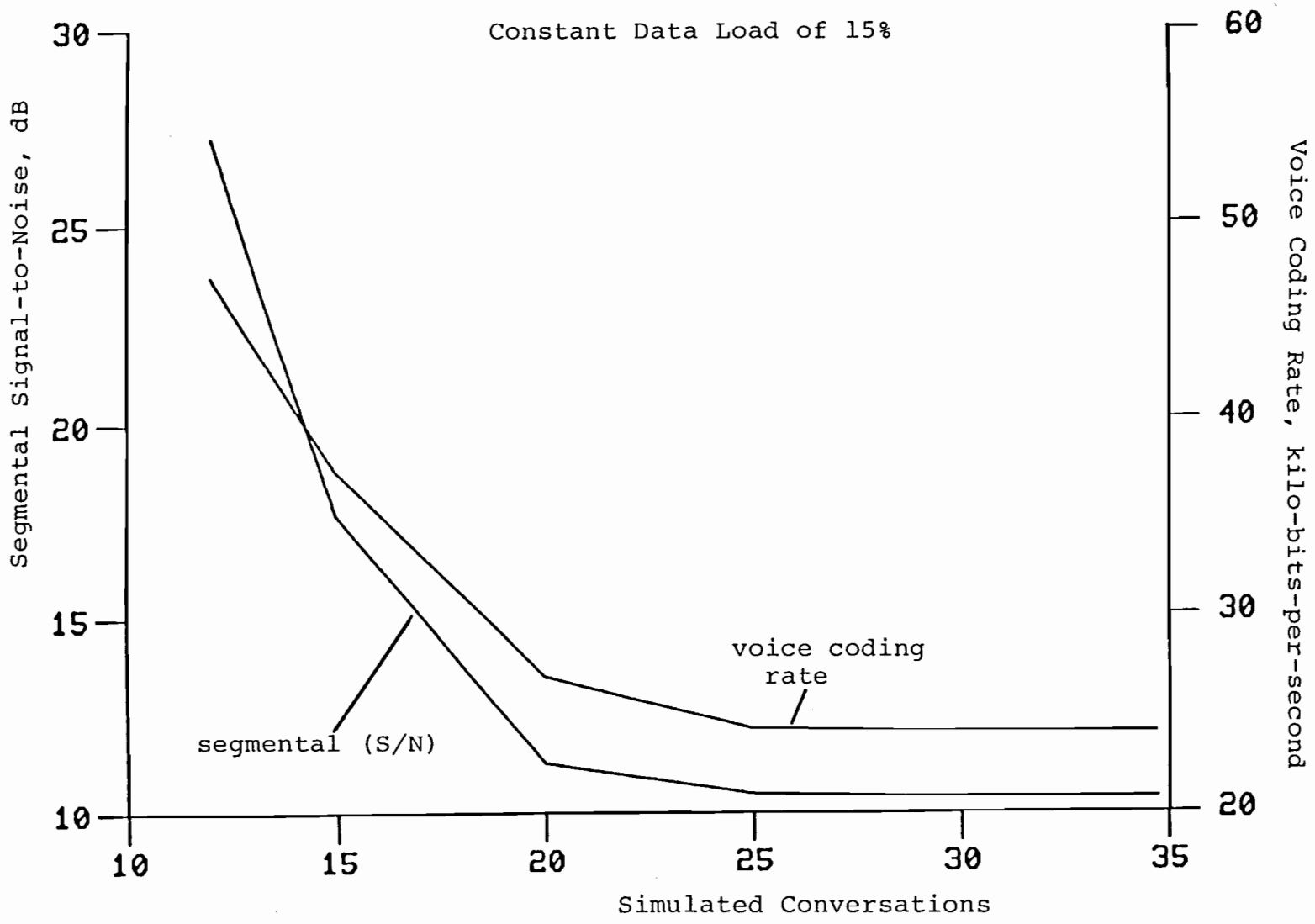


Figure 4-19. Comparison of voice coding rate in kilo-bits-per-second and the segmental signal-to-noise in decibels.

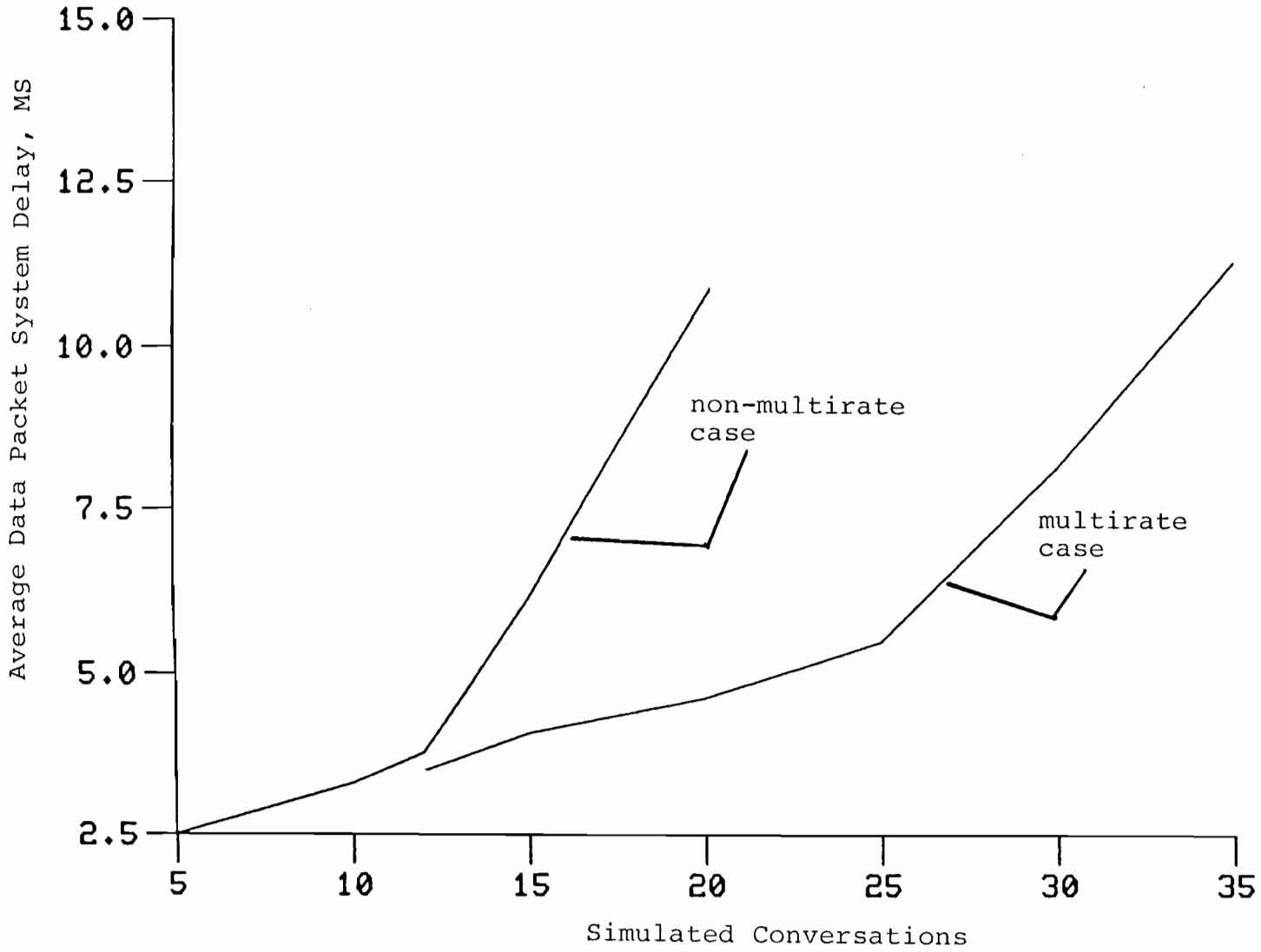


Figure 4-20. Comparison of the total system delay of data packets for the multirate and non-multirate voice coding cases.

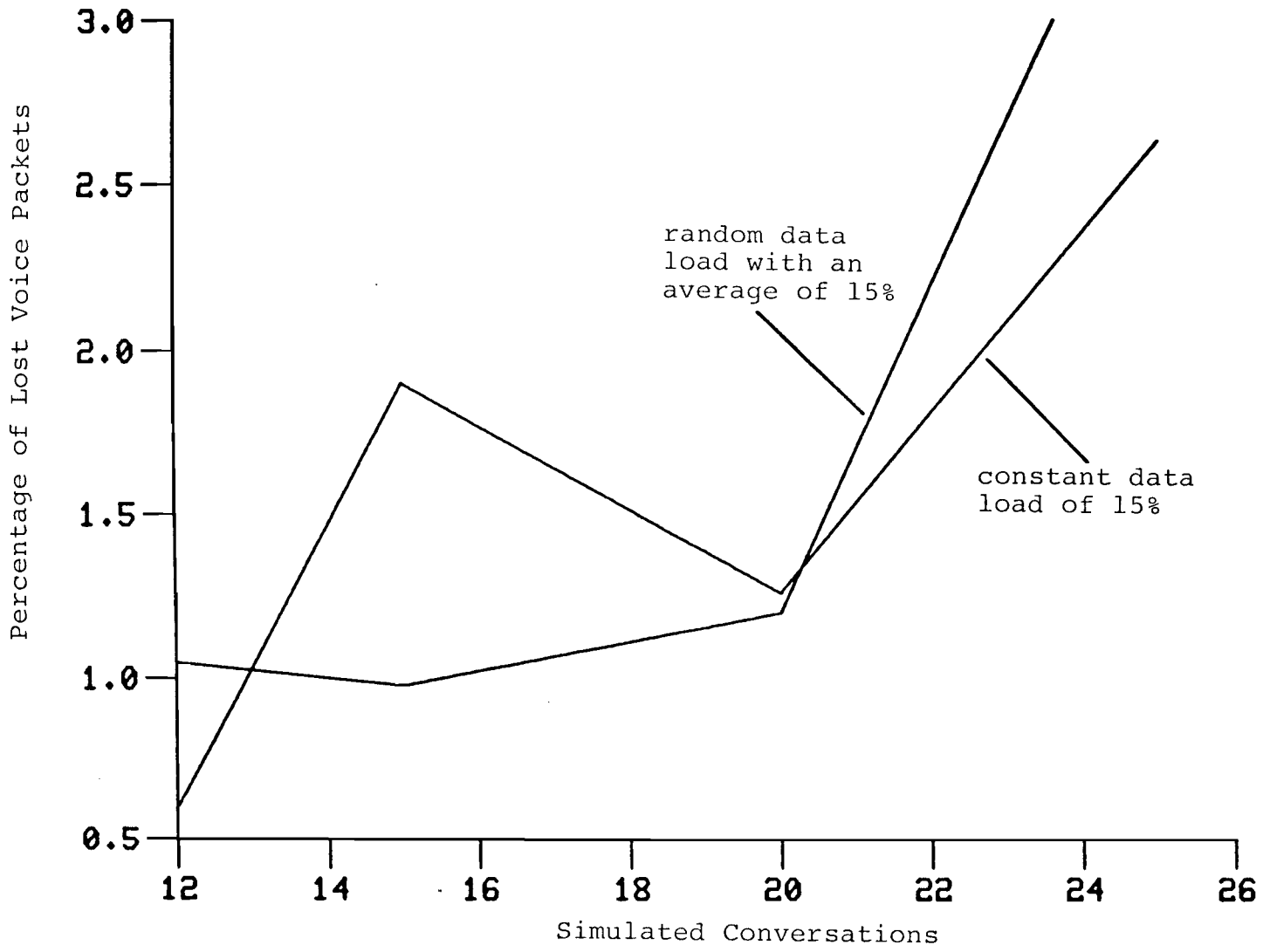


Figure 4-21. Comparison of the percentage of lost voice packets for a constant and random data load.

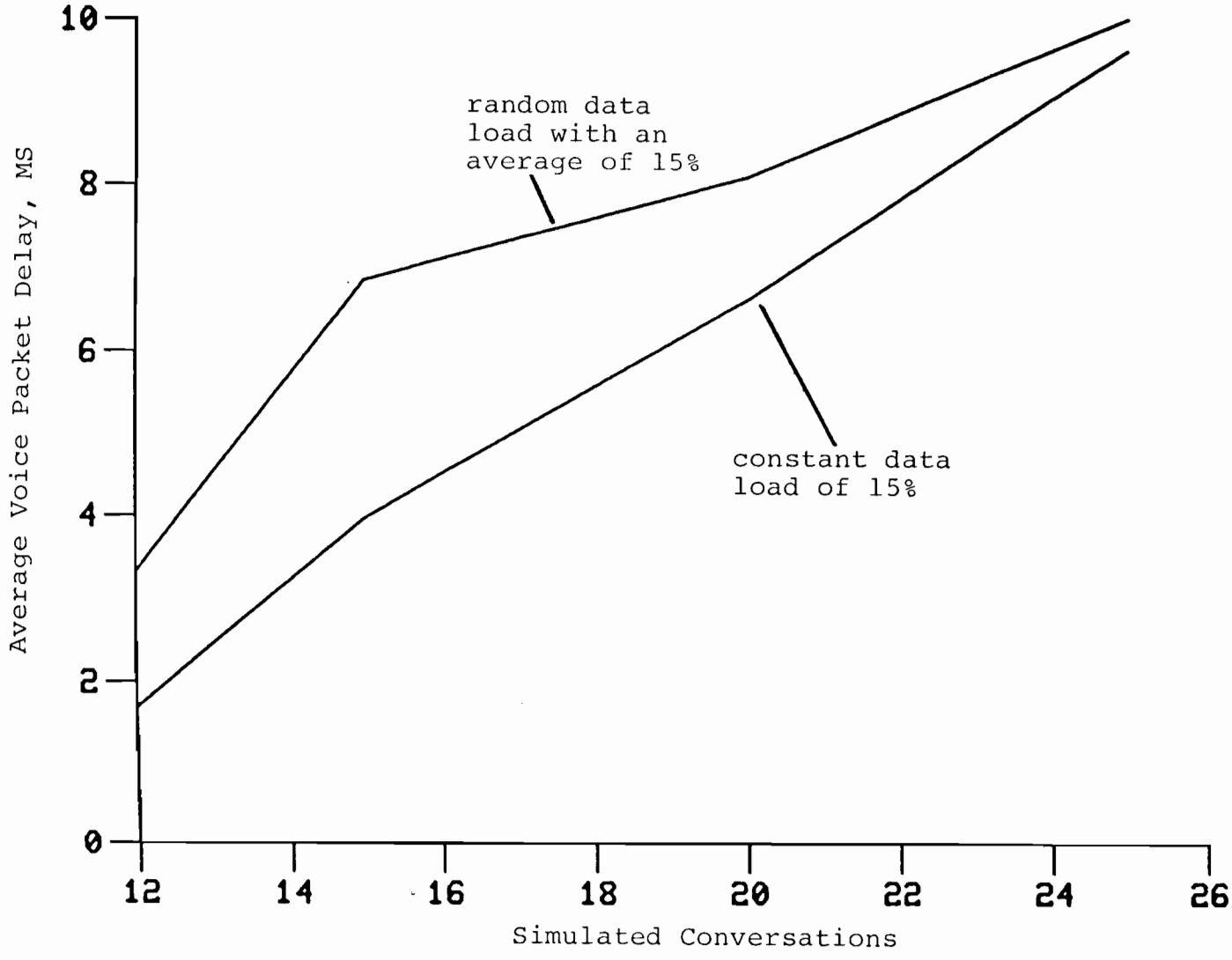


Figure 4-22. Comparison of the average voice packet delay in milliseconds for a constant and random data load.

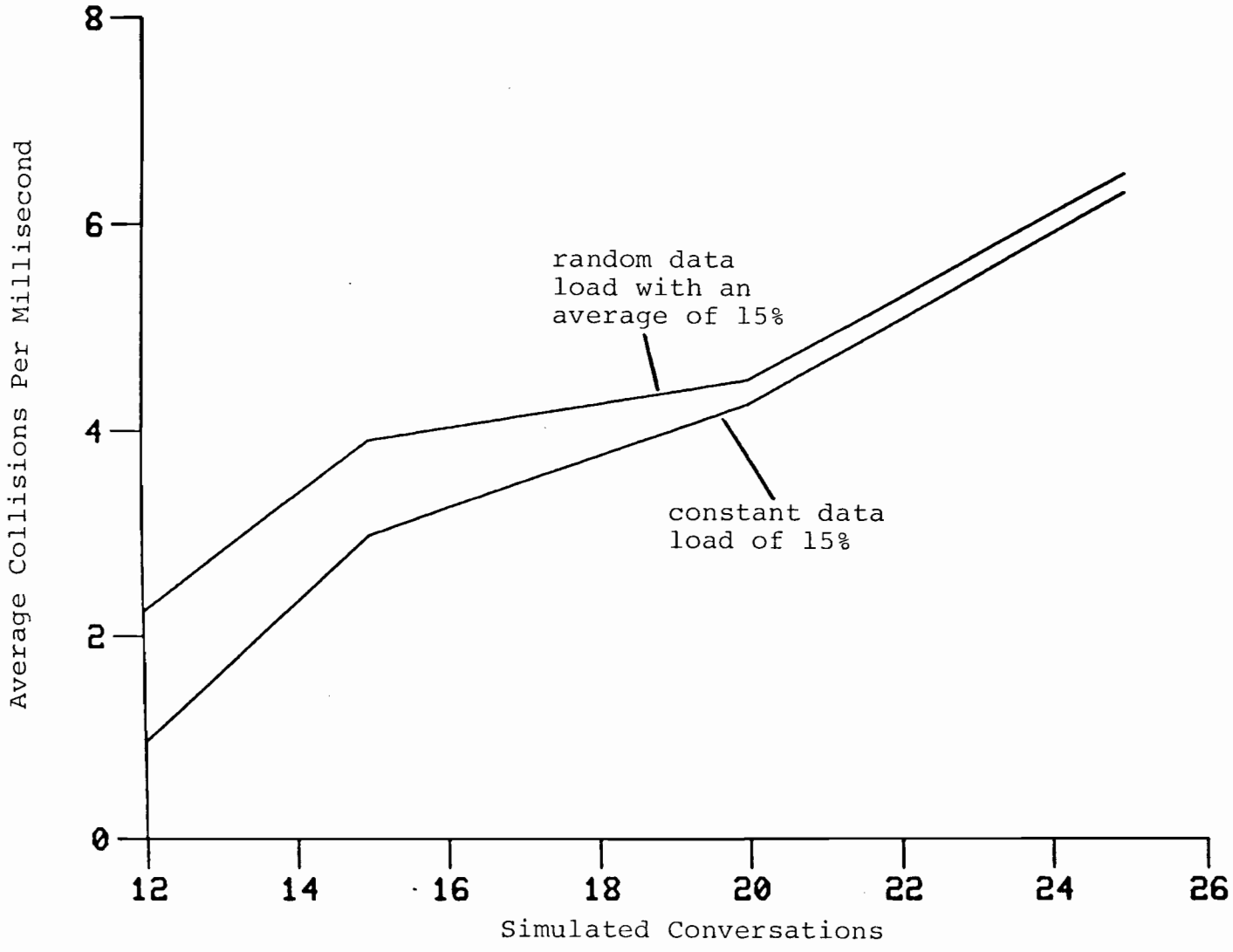


Figure 4-23. Comparison of the collisions per millisecond measurement for a constant and random data load.

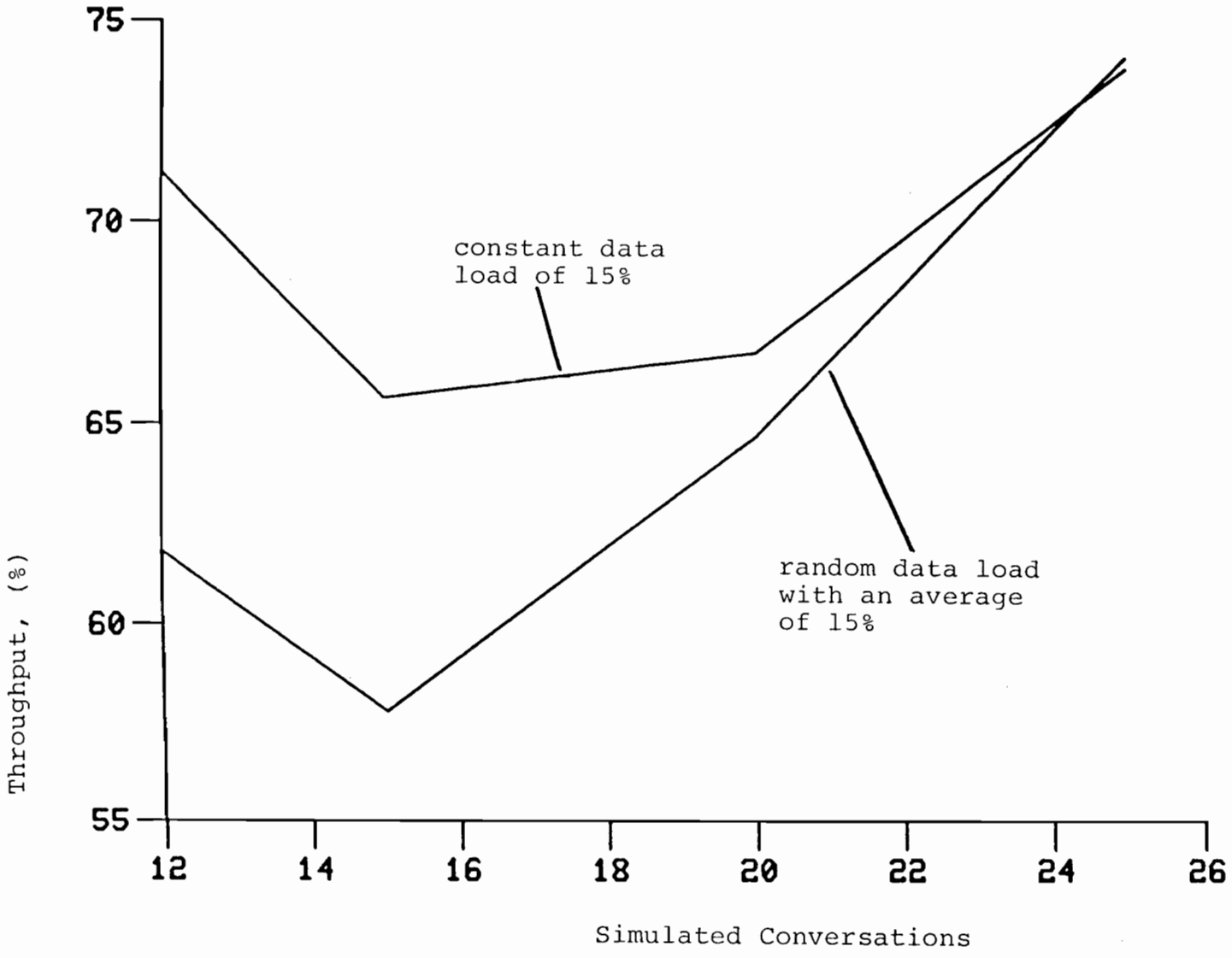


Figure 4-24. Comparison of the throughput for a constant and random data load.

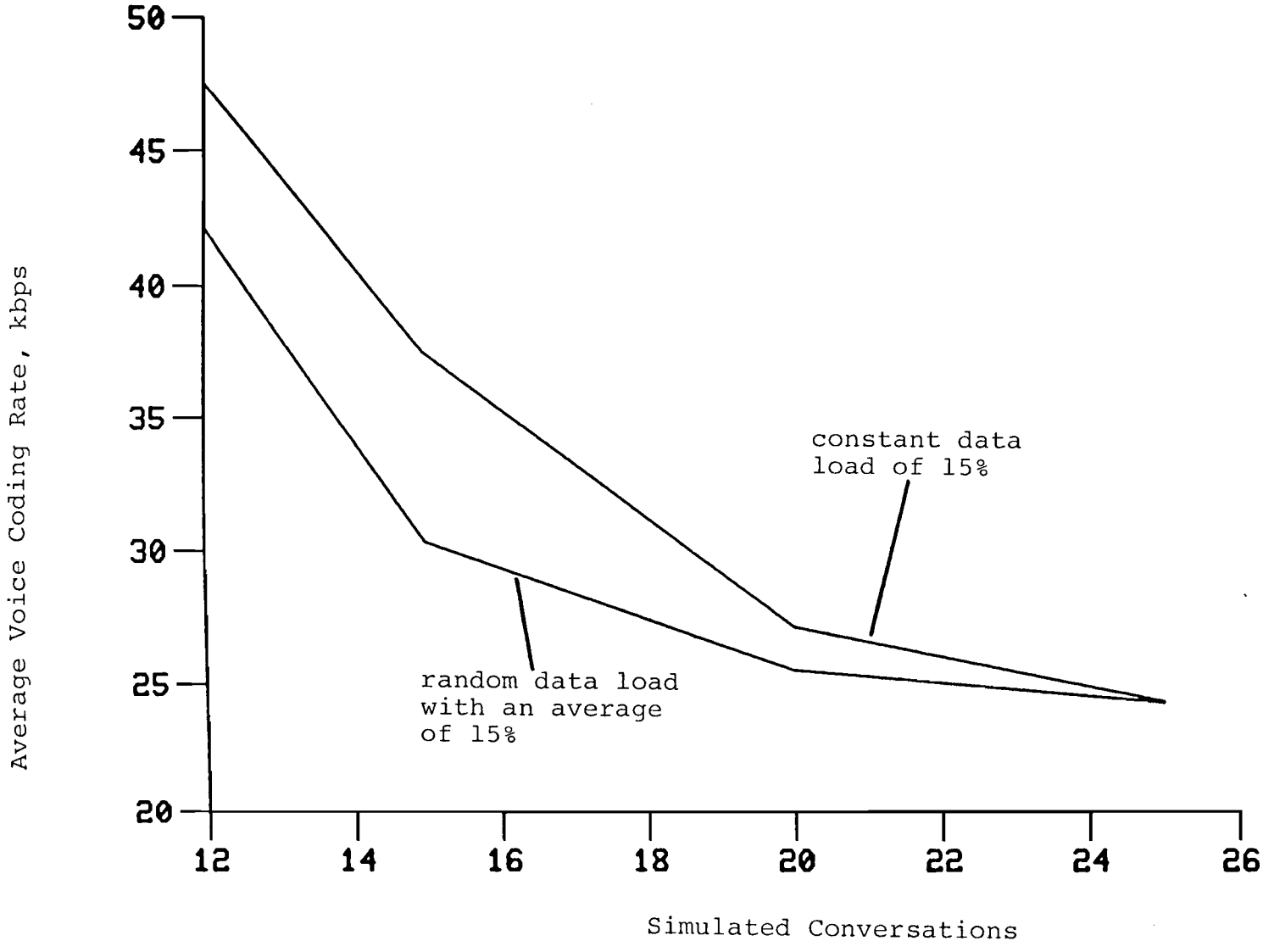


Figure 4-25. Comparison of the average voice coding rate in kilo-bits-per-second for a constant and random data load.

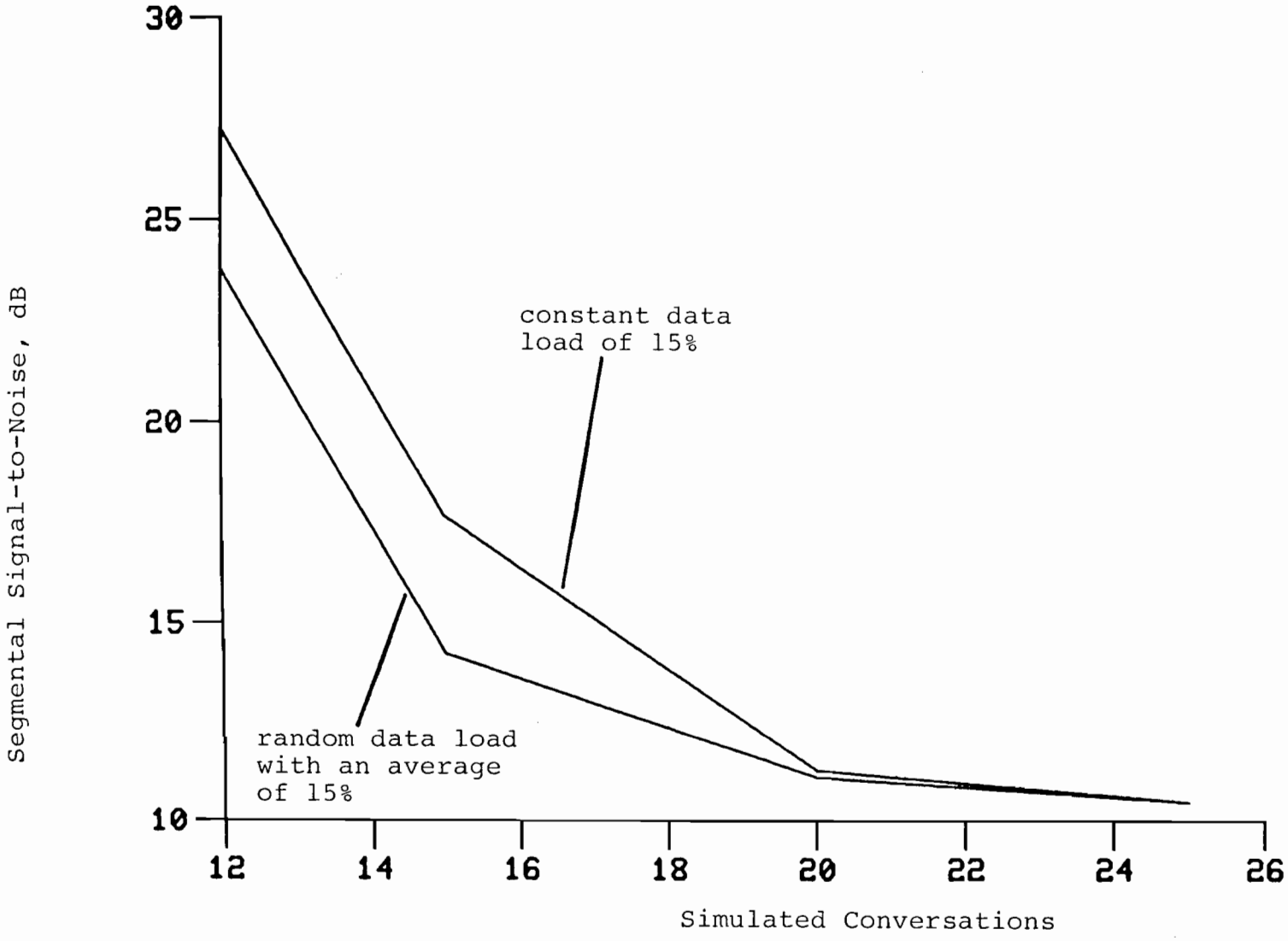


Figure 4-26. Comparison of the segmental signal-to-noise for a constant and random data load.

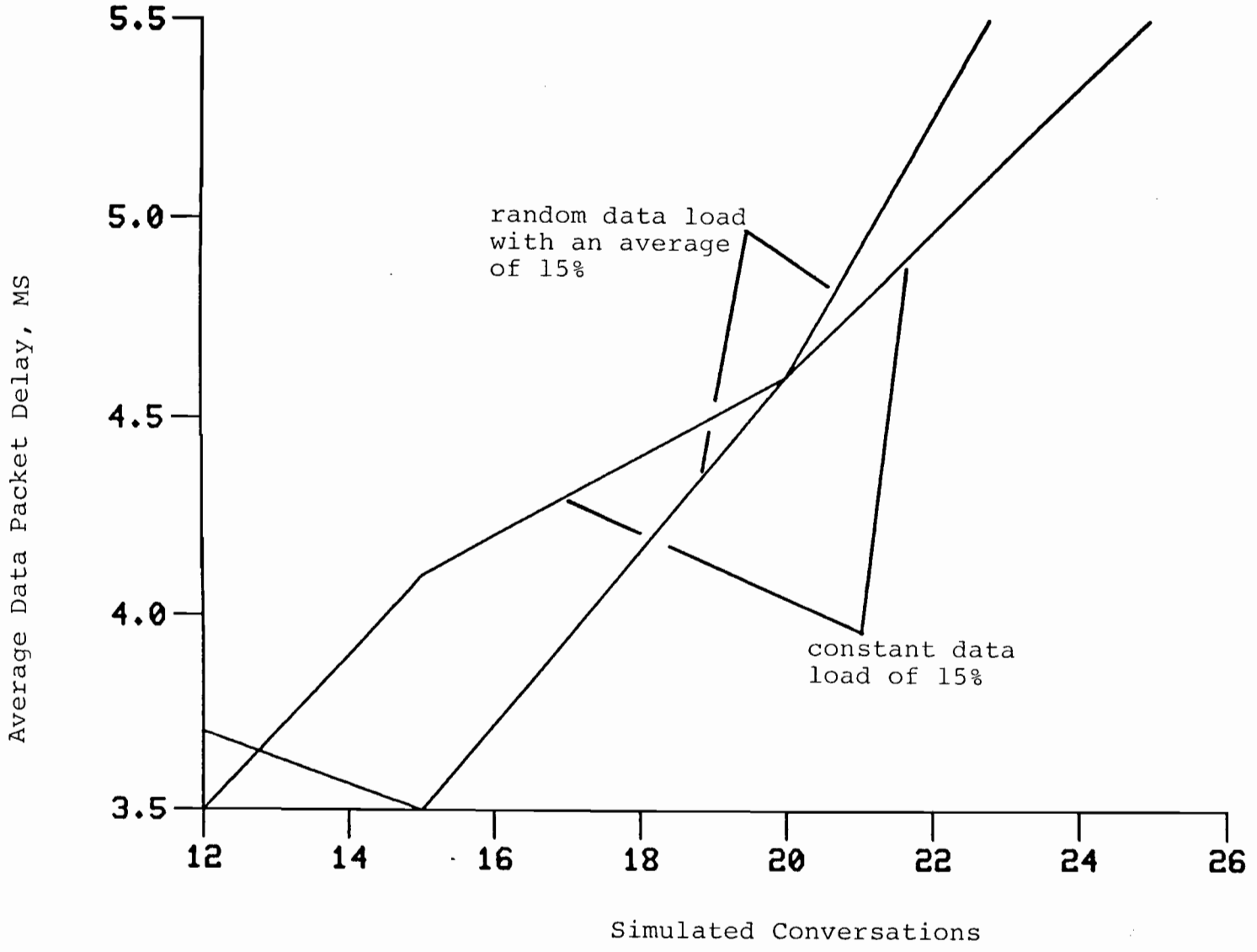


Figure 4-27. Comparison of the average total system delay of data packets for a constant and random data load.

The figures show that the random load case closely resembles the constant load case. The random load case is simulating bursty data traffic. Bursty traffic is expected on data networks. The results show that the system holds up under bursty traffic.

4.6 System Dynamics

In the previous section the simulation results were presented. Those results gave the system performance as an average. The time response of the system is useful for verifying that the feedback algorithm is operating properly. In this section, the time response of the system will be given. That is, the voice coding rate and the number of collisions per millisecond will be presented as a function of time. The time response of the system is presented as a series of figures, the voice coding rate versus time and the collisions per millisecond versus time are presented for several system configurations. All the results presented pertain to the multirate voice coding case, and have been chosen from the constant data load case and the random data load case.

The random data load case shows how the feedback system adapts to changes in the network load. That is, when the data load changes the voice coding rate is seen to change according to the amount of data traffic. The random data load case is presented for 12 voice conversations.

For the constant data load case the time response is presented for an increasing number of simulated conversations. The constant data load case shows that the system adapts to increasing voice traffic. The voice coding rate is seen to decrease when the number of simulated conversations is increased.

In Figure 4-28 the voice coding rate is shown as a function of time, the data load is a constant 15% and 12 conversations are simulated. The average

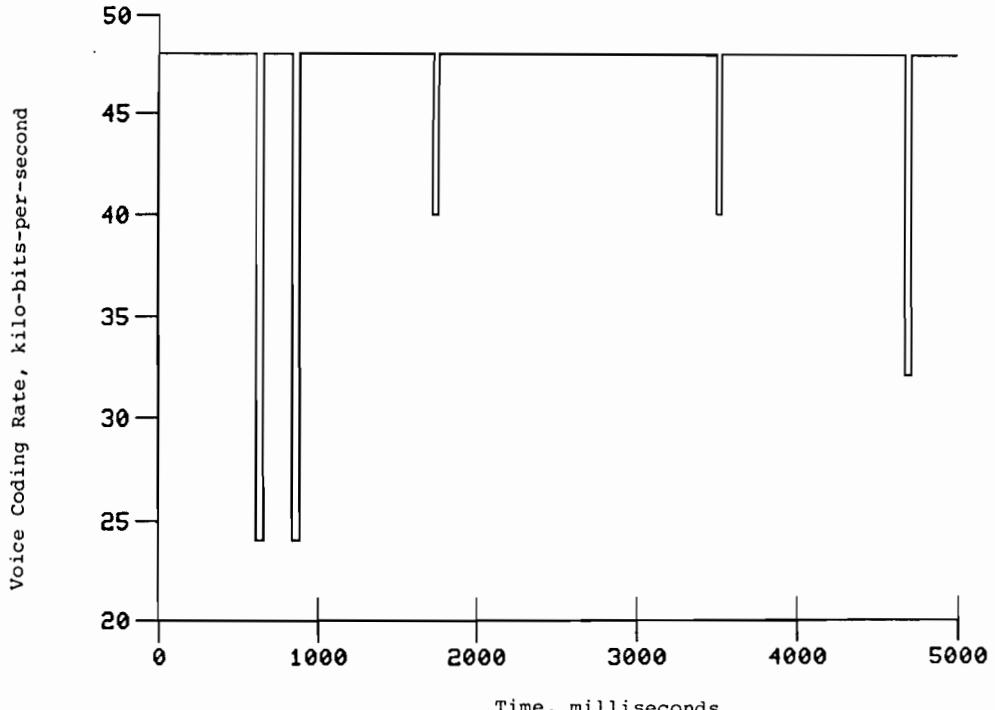


Figure 4-28. Voice coding rate versus time, for 12 simulated conversations and a constant data load of 15%.

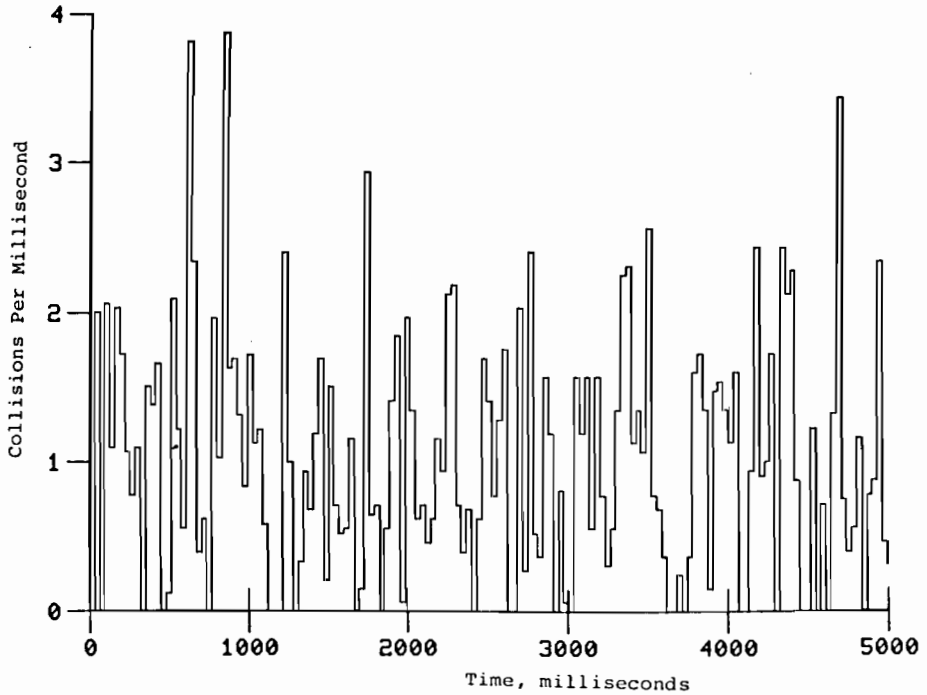


Figure 4-29. Collisions per millisecond versus time, for 12 simulated conversations and a constant data load of 15%.

voice coding rate over this period is 47.5 kbps. The number of collisions per millisecond as a function of time is shown in Figure 4-29, an average of approximately one collision occured every millisecond.

The next two figures, Figure 4-30 and Figure 4-31, show the voice coding rate and collisions per millisecond when the number of conversations is increased to 15. The data load here is also a constant 15%. The number of collisions per millisecond has increased, the average in Figure 4-31 is 2.7. An average voice coding rate of 39.0 kbps is shown in Figure 4-30. The voice coding rate is seen to vary throughout this time range.

In Figure 4-32 and Figure 4-33, the coding rate and collisions per millisecond are shown, respectively. For these figures the data load is a constant 15% and there are 20 conversations being simulated. It is seen that the coding rate falls to the lower values and remains at the lower coding rates. The corresponding collisions per millisecond remain high. The average coding rate is 30.4, and there is an average of 3.9 for the collisions per millisecond.

The final constant data load case is shown in Figure 4-34 and Figure 4-35, where 25 conversations are simulated. This case shows that at high loads the voice coding rate remains at the lowest value. The average coding rate here is 25.5 kbps. The corresponding collisions per millisecond are higher than the previous cases, an average of 5.8 is shown.

The remaining portion of this section is concerned with the random data load case. The figures presented show the voice coding rate and collisions per millisecond versus time. The data load changes at 5000 milliseconds in each of the figures. It can be seen from these figures that the feedback algorithm adapts to the changing load conditions. The results are given for the case where 12 conversations are simulated. These results are presented in Figure 4-36 through Figure 4-43.

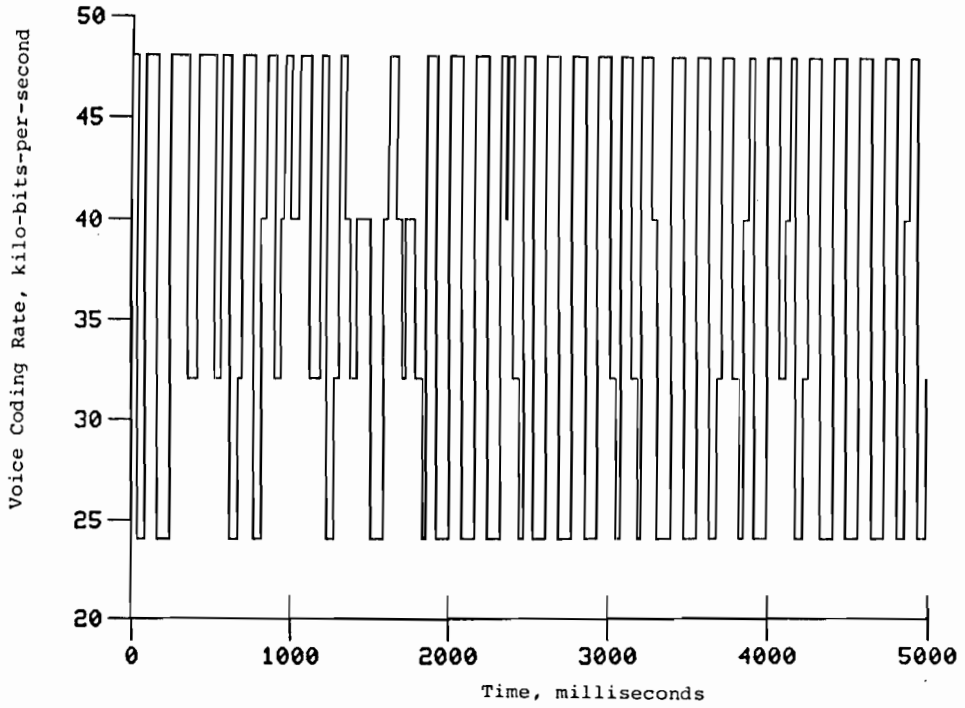


Figure 4-30. Voice coding rate versus time, for 15 simulated conversations and a constant data load of 15%.

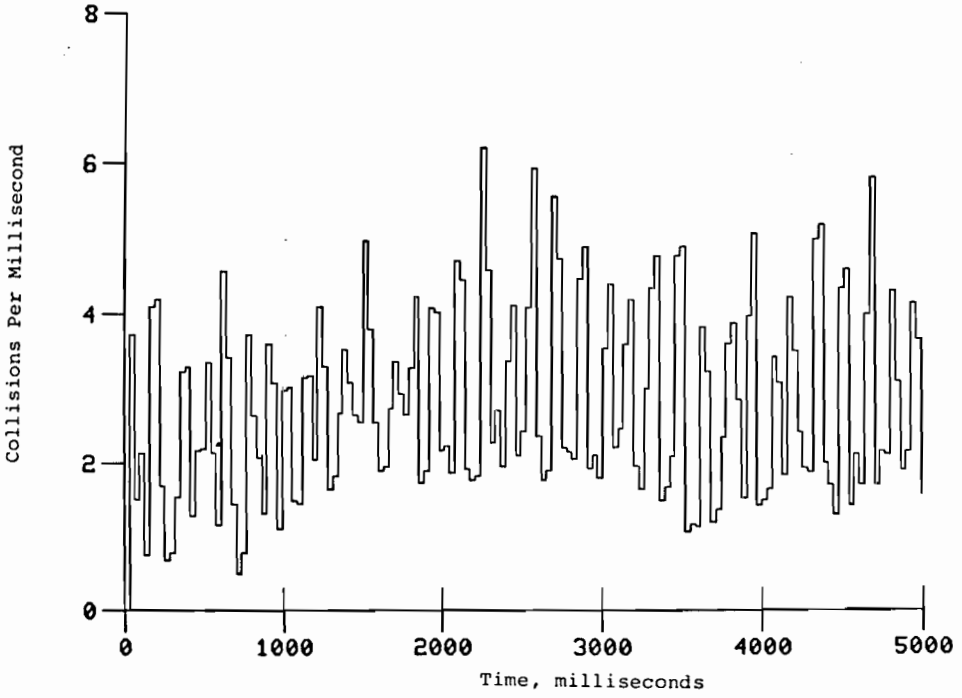


Figure 4-31. Collisions per millisecond versus time, for 15 simulated conversations and a constant data load of 15%.

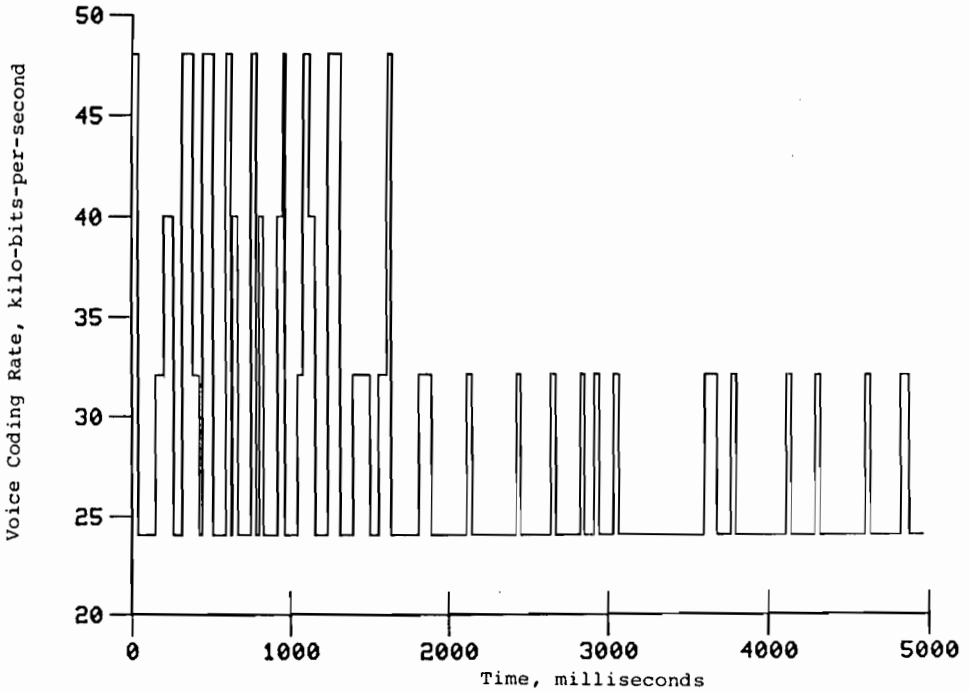


Figure 4-32. Voice coding rate versus time, for 20 simulated conversations and a constant data load of 15%.

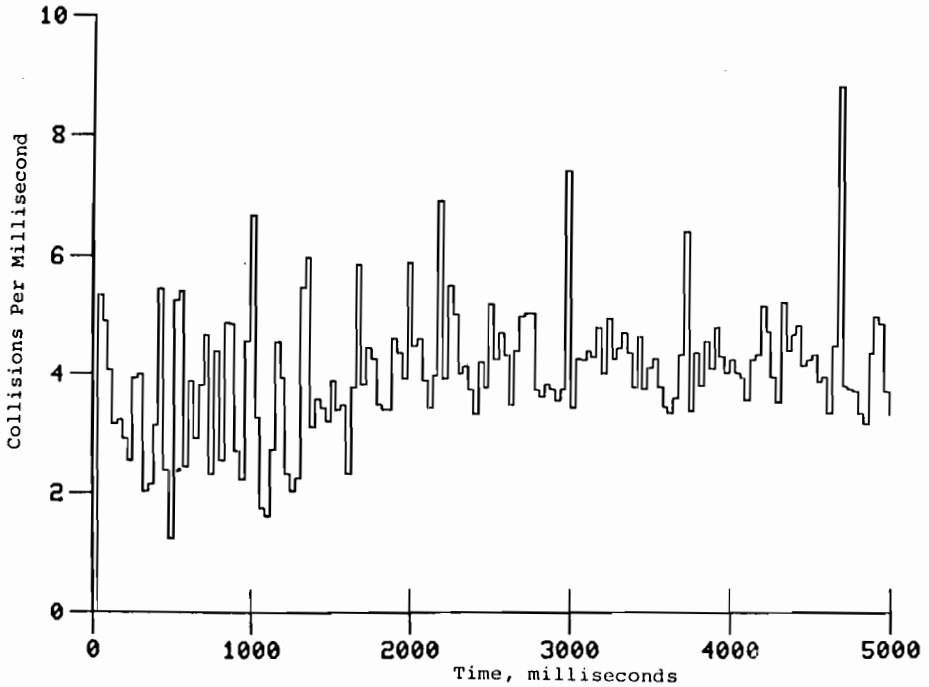


Figure 4-33. Collisions per millisecond versus time, for 20 simulated conversations and a constant data load of 15%.

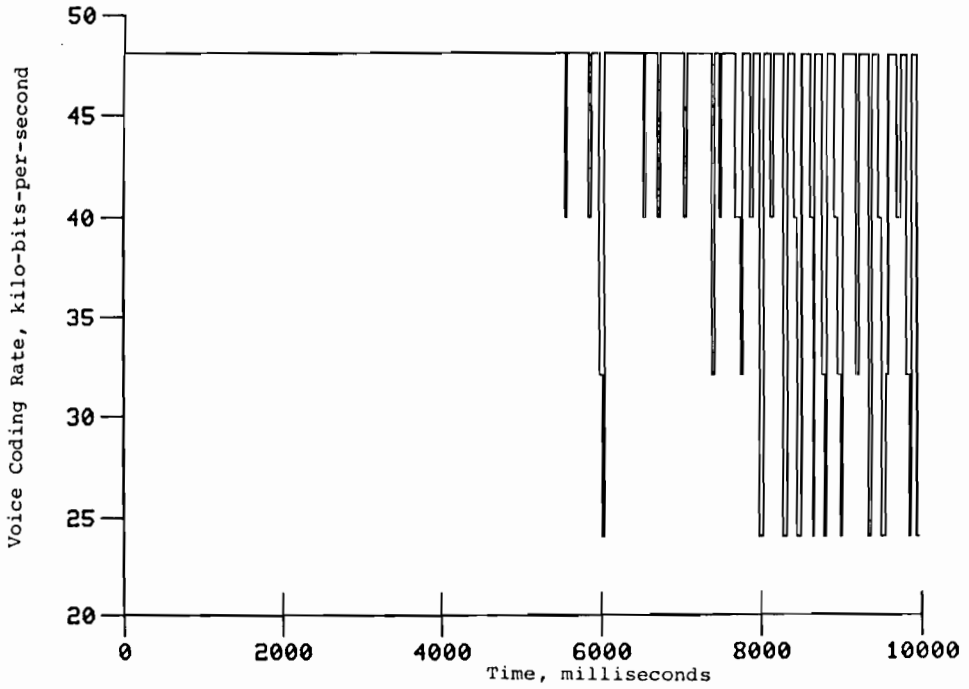


Figure 4-36. Voice coding rate versus time, for 12 simulated conversations and a data load that changes from 5% to 20% at 5000 milliseconds.

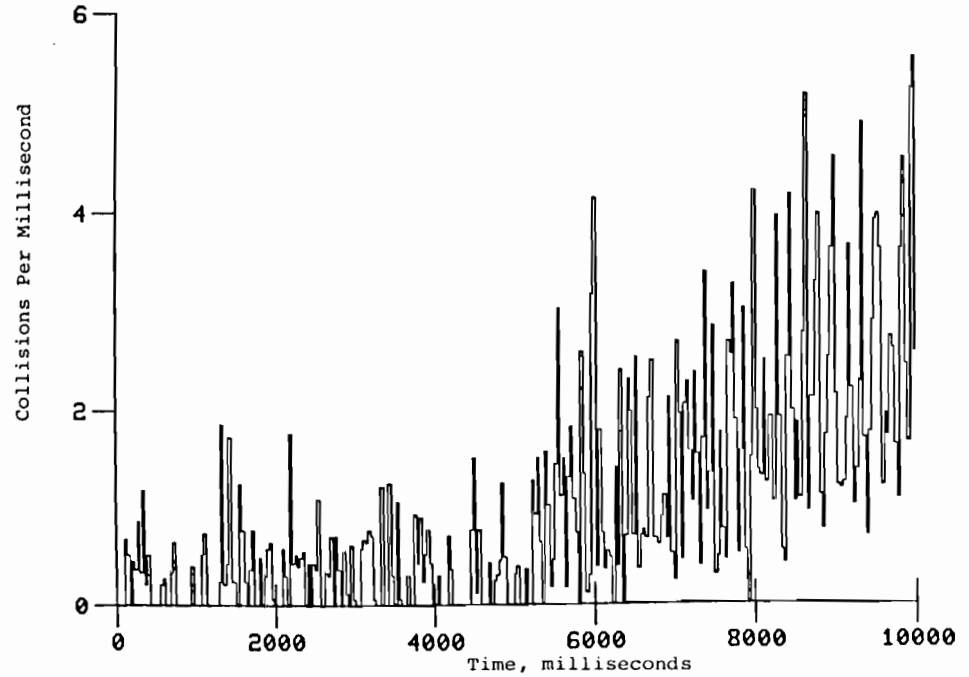


Figure 4-37. Collisions per millisecond versus time, for 12 simulated conversations and a data load that changes from 5% to 20% at 5000 milliseconds.

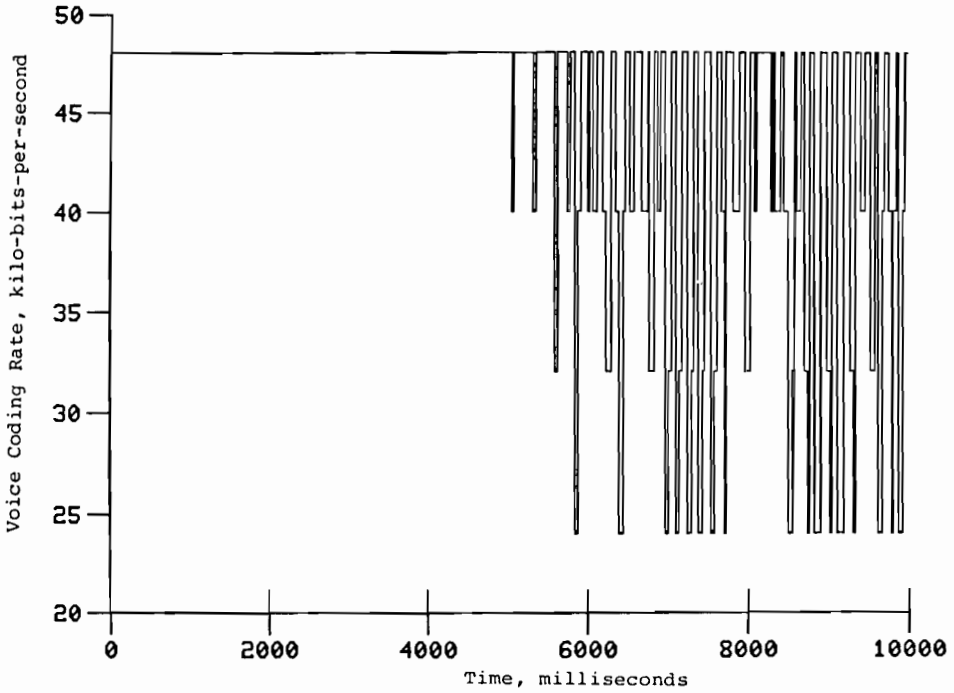


Figure 4-38. Voice coding rate versus time, for 12 simulated conversations and a data load that changes from 10% to 25% at 5000 milliseconds.

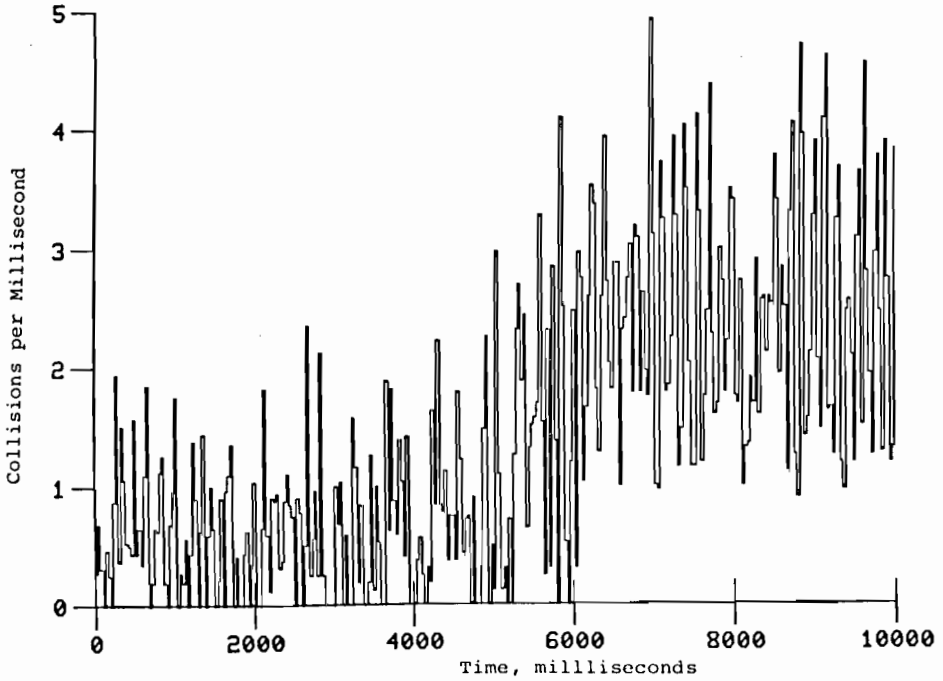


Figure 4-39. Collisions per millisecond versus time, for 12 simulated conversations and a data load that changes from 10% to 25% at 5000 milliseconds.

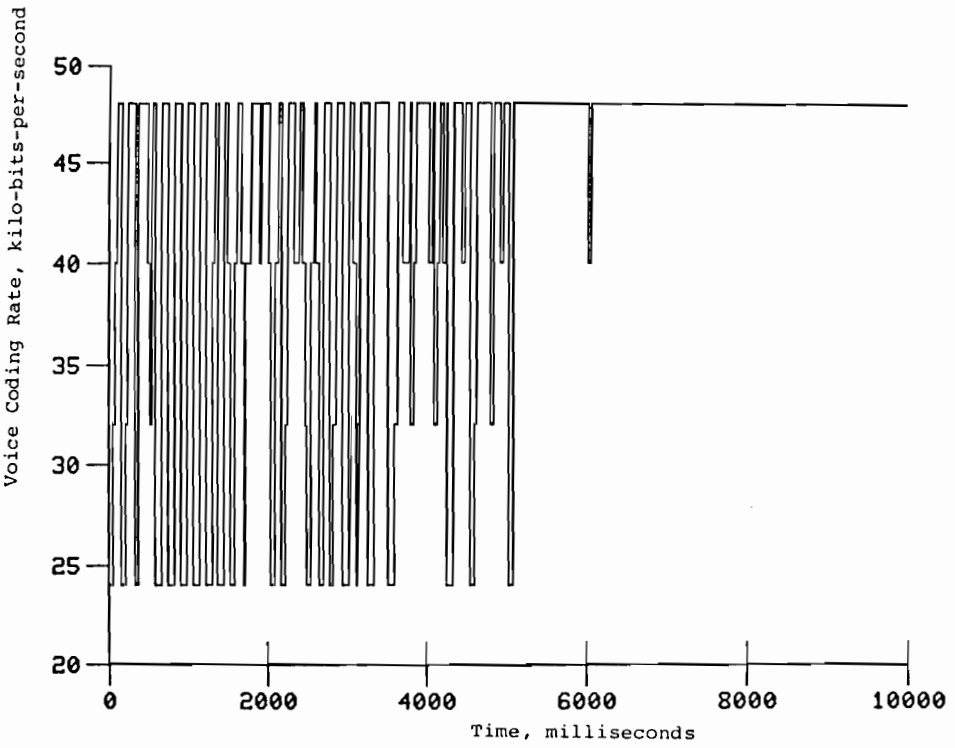


Figure 4-40. Voice coding rate versus time, for 12 simulated conversations and a data load that changes from 25% to 5% at 5000 milliseconds.

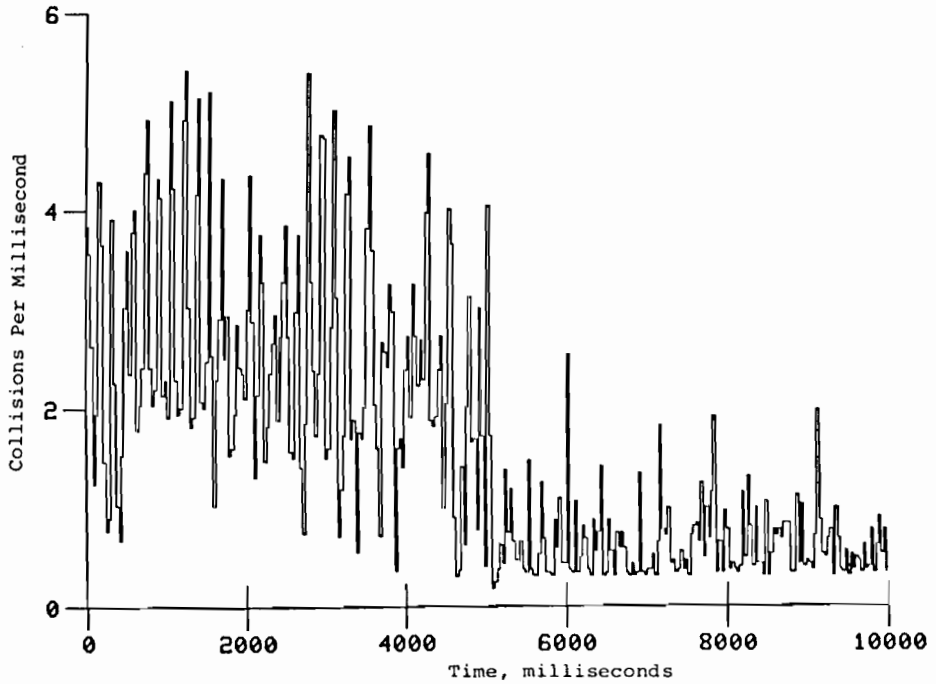


Figure 4-41. Collisions per millisecond versus time, for 12 simulated conversations and a data load that changes from 25% to 5% at 5000 milliseconds.

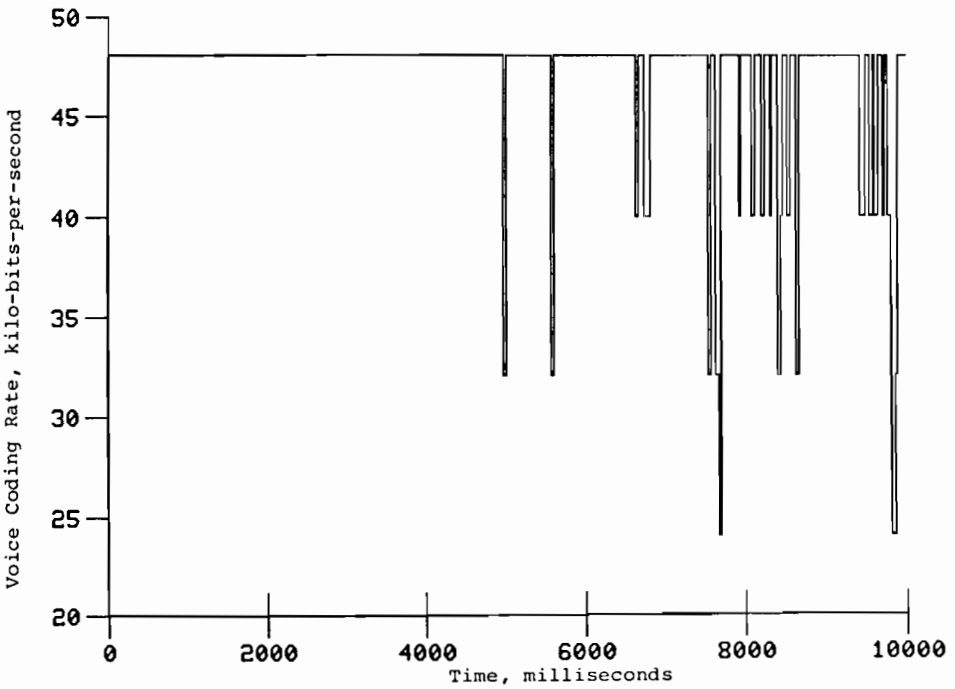


Figure 4-42. Voice coding rate versus time, for 12 simulated conversations and a data load that changes from 5% to 15% at 5000 milliseconds.

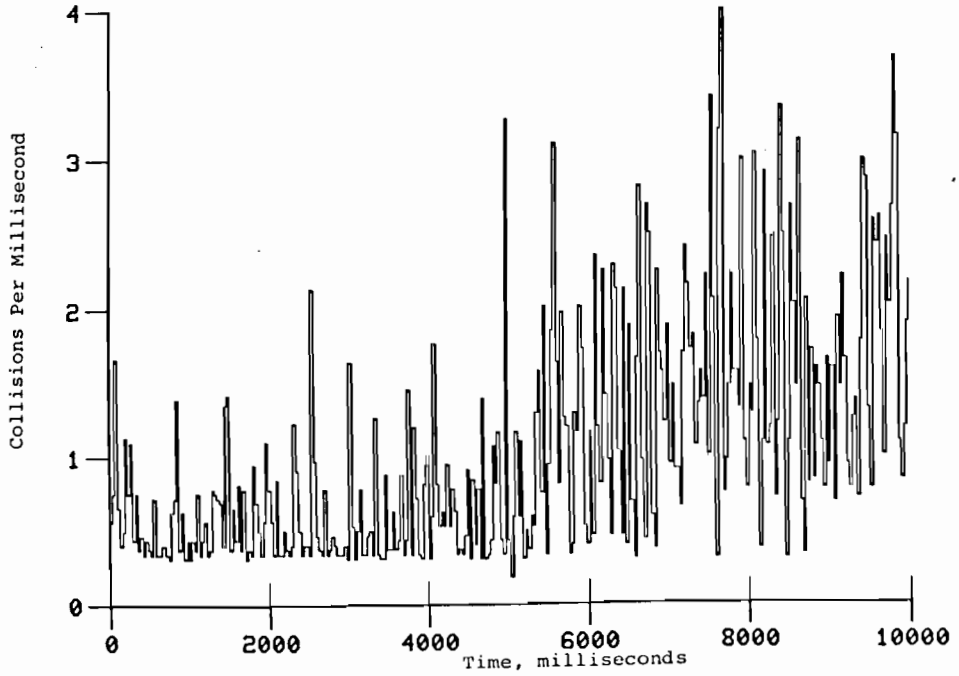


Figure 4-43. Collisions per millisecond versus time, for 12 simulated conversations and a data load that changes from 5% to 15% at 5000 milliseconds.

In this section the time response of the system was presented. This information is useful for verifying that the feedback algorithm is operating properly. It was shown that the system adapts to the changing load conditions.

This chapter presented the multirate voice coding algorithm. The study was done as a proof of concept. The results show that the use of multirate voice coding can improve the network performance by allowing a greater number of conversations. The multirate techniques increase the complexity of the network. The added complexity comes from the need to measure the traffic on the network and the multirate voice coder.

The choice of voice coding rates was limited to the constraints of the particular coder that was implemented at the Telecommunications and Information Sciences Laboratory. It has been suggested that the voice coding rates should be chosen in increments no larger than 2 kbps [20]. However, the goal here was to prove, using a simplified system, that the multirate techniques can improve the performance of CSMA/CD networks that have a combined loading of voice and data.

5.0 CONCLUSION

The CSMA/CD simulation model, documented in Chapter 3, accurately predicts the performance of a real system. The simulation model was developed so that studies of Ethernet-like networks could be performed. The specific Local Area Network (LAN) configuration studied has a combined loading of voice and data. The objective of the study was to determine the feasibility in using a multirate voice coding scheme to control the network load.

The multirate voice coding system performs well under varying load conditions. A comparison was made between the multirate voice coding system and a voice/data network that uses a constant voice coding rate. The results of that comparison show that the multirate system gives a superior performance. Specifically, an increase of ten conversations is achieved when the variable rate coding scheme is applied to a system that consists of a 1Mbps line and a constant data load of 15%.

In addition, the constant data load and random data load cases were compared. This comparison shows that the constant load case closely approximates the random load case. Of more significance, the random load case simulates the bursty traffic associated with data. Therefore, the multirate coding techniques perform well under bursty traffic.

6.0 REFERENCES

[1] A. Alan and B. Pritsker, "Introduction to Simulation and SLAM II," Wiley, 1984.

[2] Ed Friedman, "Introduction to SLAM - Seminar Notes," TISL Technical Memo 6731-1, February, 1985.

[3] Robert M. Metcalf and David R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Communications of the ACM, Volume 19, Number 7, July 1976.

[4] John F. Shoch, Yogen K. Dalal, David D. Redell, and Ronald C. Crane, "Evolution of the Ethernet Local Computer Network," IEEE Computer Society Magazine, August 1982, pp. 10-27.

[5] The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications, Version 2.0, Digital Equipment Corporation, Intel, Xerox, November 1982.

[6] John F. Shoch, "Design and Performance of Local Computer Networks," Ph.D. Dissertation, Dept. of Computer Science, Stanford University, 1979.

[7] John F. Shoch and Jon A. Hupp, "Performance of an Ethernet Local Network -- A Preliminary Report," COMPCON Spring 1980, pp. 318-322.

[8] --------, "Measured Performance of an Ethernet Local Network," Communications of the ACM, Volume 23, Number 12, December 1980, pp. 711-721.

[9] Herman D. Hughes and Liang Li, "A Simulation Model of the Ethernet," Dept. of Computer Science, Michigan State University East Lansing, Technical Report TR 82-008, 1982.

[10] A.S. Acampora, M.G. Hluchyj, and C.D. Tsao, "Performance of a Centralized-bus Local Area Network," Internal Report, Engineering Design and Development, American Bell.

[11] A.S. Tanenbaum, "Computer Networks," Prentice-Hall, 1981.

[12] J. D. DeTreville, "A Simulation-Based Comparison of Voice Transmission on CSMA/CD Networks and on Token Buses," AT&T Bell Laboratories Technical Journal, Vol. 63, No. 1, January 1984, pp. 33-55.

[13] G. J. Nutt and D. L. Bayer, "Performance of CSMA/CD Networks Under Combined Voice and Data Loads," IEEE Trans. on Communications, Vol. COM-30, No. 1, January 1982, pp. 6-11.

[14] J. M. Musser, T. T. Liu, L. Li, and G. J. Boggs, "A Local Area Network as a Telephone Subscriber Loop," IEEE Journal on Selected Areas in Communications, Vol. SAC-1, No. 6, December 1983, pp. 1046-1054.

[15] N. S. Jayant and S. W. Christensen, "Effects of Packet Losses in Waveform Coded Speech and Improvements Due to an Odd Even Sample-Interpolation Procedure," IEEE Trans. on Communications, Vol. COM-29, No. 2, February 1981, pp. 101-109.

[16] H. M. Heggestad, R. J. McAulay, and J. Tiernay, "Practical Considerations for Speech Digitizing Systems at Rates from 64.0 to 0.6 kbps," IEEE Global Telecommunications Conf., Miami, Florida, November 1982, pp. 349-356.

[17] D. J. Goodman, "Embedded DPCM for Variable Bit Rate Transmission," IEEE Trans. on Commun., Vol. COM-28, No. 7, July 1980, pp. 1040-1046.

[18] N. S. Jayant and P. Noll, "Digital Coding of Waveforms," Prentice-Hall, 1984.

[19] M. Ilyas and H. T. Mouftah, "Performance Evaluation of Computer Communications Networks," IEEE Communications Magazine, Vol. 23, No. 4, April 1985, pp. 18-29.

[20] S. Seneff, "Computer Simulation Model for a Digital Communications Network Utilizing an Embedded Speech Encoding Technique," M.I.T. Lincoln Lab., Lexington, MA, Tech. Note 1978-33, DDC# AD-A065182, October, 1978.

APPENDIX A LISTING OF THE CSMA/CD SIMULATION MODEL

Appendix A.1 Listing of the Network Model

```
GEN,TISL,CSMA CD,8/20/84,1,NO,NO;
LIMITS,24,17,200;
STAT,1,NODE 1 SYS. TIME
STAT,2,NODE 2 SYS. TIME
STAT,3,NODE 3 SYS. TIME
STAT,4,NODE 4 SYS. TIME
STAT,5,NODE 5 SYS. TIME
STAT,6,NODE 6 SYS. TIME
STAT,7,NODE 7 SYS. TIME
STAT,8,NODE 8 SYS. TIME
STAT,9,NODE 9 SYS. TIME
STAT,10,NODE 10 SYS TIME
STAT,11,NODE 11 SYS TIME
STAT,12,NODE 12 SYS TIME
STAT,13,NODE 13 SYS TIME
STAT,14,NODE 14 SYS TIME
STAT,15,NODE 15 SYS TIME
;
STAT,16,QUEUEING DELAY
STAT,17,ACCESS DELAY
STAT,18,CHANNEL DELAY
STAT,19,TOTAL SYS DELAY
;
NETWORK;
;
;       VARIABLE DEFINITIONS:
;
;         atrib(1) = packet creation time
;         atrib(2) = node identification
;         atrib(3) = propagation left marker, if left prop is
;                    finished then atrib(3)=0
;         atrib(4) = propagation right marker, if right prop is
;                    finished then atrib(4)=0
;         atrib(5) = if atrib(5)=0 then event being scheduled is propagation,
;                    if atrib(5)=1 then event being scheduled is transmission
;         atrib(6) = counter for the number of attempts to transmit
;         atrib(7) = end of propagation left marker, if atrib(7)=0
;                    then finished prop left
;         atrib(8) = end of propagation right marker, if atrib(8)=0
;                    then finished prop right
;         atrib(9) = packet length in microseconds
```

```
;       atrib(10)= atrib(9)-slttim = amount of time left after the
;                  collision discrimination period
;       atrib(11)= number of bits per packet
;       atrib(12)= marks the time transmission is attempted, gets
;                  reset each time a retransmission is attempted
;       atrib(13)= while packet is in the AWAIT NODE atrib(13)=0, set
;                  to 1 when packet leaves the queue, used to collect
;                  statistics on the amount of time a packet is queued,
;                  tqueud= atrib(14) minus atrib(1)if atrib(13)=0
;       atrib(14)= marks the time when a packet leaves the node queue
;                  (the AWAIT), used to collect statistics on the time
;                  required  to access the network,
;                  taccss=atrib(12)-atrib(14)
;       atrib(15)= not used
;       atrib(16)= not used
;       atrib(17)= first access attribute, if the packet is trying to
;                  transmit for the first time then atrib(17) is set
;                  to 1 from 0, on subsequent attempts to transmit
;                  atrib(17) is incremented
;
;     XX(25) = THE SIMULATION TIME (TTFIN)
;
;       files  1-15 reserved for the AWAIT NODE at each station
;       files 16-21 not used
;       file    22  channel QUEUE NODE
;       file    23  not used
;       file    24  defer file
;       file    25  event calendar
;
      RESOURCE/PACK1,1/PACK2,2/PACK3,3/PACK4,4/PACK5,5/PACK6,6/
               PACK7,7/PACK8,8/PACK9,9/PACK10,10/PACK11,11/
               PACK12,12/PACK13,13/PACK14,14/PACK15,15;
;
;
;
;     MODEL OF THE NUMBER OF STATIONS ON THE ETHER
;     ============================================
;
;-------------------------------------------------------------------------
;
      CREATE,EXPON(13931.97265625),,1;create packets at node 1
      ASSIGN,ATRIB(2)=1.0,
             ATRIB(9)=1393.197265625,
             ATRIB(11)=4096.0,1; assign attributes
      AWAIT(1),PACK1;          packets wait for previous packet to finish
      ACT,,,CHNL;              immediate branch to the channel
;
;-------------------------------------------------------------------------
```

Appendix A.1 Continued

```
;-------------------------------------------------------------------------
;
       CREATE,EXPON(13931.97265625),1500,1;
       ASSIGN,ATRIB(2)=2.0,
              ATRIB(9)=1393.197265625,
              ATRIB(11)=4096.0,1;
       AWAIT(2),PACK2;
       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
       CREATE,EXPON(13931.97265625),3000,1;
       ASSIGN,ATRIB(2)=3.0,
              ATRIB(9)=1393.197265625,
              ATRIB(11)=4096.0,1;
       AWAIT(3),PACK3;
       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
       CREATE,EXPON(13931.97265625),4500,1;
       ASSIGN,ATRIB(2)=4.0,
              ATRIB(9)=1393.197265625,
              ATRIB(11)=4096.0,1;
       AWAIT(4),PACK4;
       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
       CREATE,EXPON(13931.97265625),6000,1;
       ASSIGN,ATRIB(2)=5.0,
              ATRIB(9)=1393.197265625,
              ATRIB(11)=4096.0,1;
       AWAIT(5),PACK5;
       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;      CREATE,EXPON(13931.97265625),7500,1;
;      ASSIGN,ATRIB(2)=6.0,
;             ATRIB(9)=1393.197265625,
;             ATRIB(11)=4096.0,1;
;      AWAIT(6),PACK6;
;      ACT,,,CHNL;
;
;-------------------------------------------------------------------------
```

Appendix A.1 Continued

```
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),9000,1;
;       ASSIGN,ATRIB(2)=7.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(7),PACK7;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),11000,1;
;       ASSIGN,ATRIB(2)=8.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(8),PACK8;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),13000,1;
;       ASSIGN,ATRIB(2)=9.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(9),PACK9;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),15000,1;
;       ASSIGN,ATRIB(2)=10.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(10),PACK10;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),17000,1;
;       ASSIGN,ATRIB(2)=11.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(11),PACK11;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
```

Appendix A.1 Continued

```
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),20000,1;
;       ASSIGN,ATRIB(2)=12.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(12),PACK12;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),23000,1;
;       ASSIGN,ATRIB(2)=13.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(13),PACK13;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),26000,1;
;       ASSIGN,ATRIB(2)=14.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(14),PACK14;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       CREATE,EXPON(13931.97265625),29000,1;
;       ASSIGN,ATRIB(2)=15.0,
;              ATRIB(9)=1393.197265625,
;              ATRIB(11)=4096.0,1;
;       AWAIT(15),PACK15;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;       COMBINE STATIONS TO FORM CHANNEL
;       ================================
;
CHNL   QUEUE(22);                  dump entities into queue
;
       ACT;                        immediate branch to channel model
;
       EVENT,1;                    gateway to discrete event program
;
;
;
;
       ASSIGN,XX(25)=10000000;     set xx(25) equal to ttfin
       END;
INIT,0,10000000;
FIN;
```

Appendix A.2 Listing of the Discrete Event Model

```
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine event(i)
c
c short explanation:
c       The event subroutine is required by SLAM in all discrete event
c       models the user will schedule an event to occur and the event
c       subroutine calls the particular event when the time scheduled is
c       reached.
c
      go to (1,2,3,4,5,6,7,8),i
    1 call sense
      return
    2 call transmit
      return
    3 call leftprop
      return
    4 call rigtprop
      return
    5 call success
      return
    6 call endtrans
      return
    7 call ltfinprop
      return
    8 call rtfinprop
      return
      end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine sense
c
c short explanation:
c       The sense event is scheduled from the calc_wait_backoff and
c       exitdefer subroutines.  In sense statistic collection is done, and
c       statistic collection attributes are set.  Then sense looks at the
c       network status array to see if the channel is idle, if the channel
c       is idle a transmit event is scheduled, if the channel is not idle
c       the packet gets put in the defer file.
c
      include 'params.dat'
      integer inode,i
      real tqueud
      atrib(17)=atrib(17)+1
c             increment the first access atrib
      if  ( atrib(13) .eq. 0 ) then
c                   the packet has just left the node queue, so collect
c                   statistics and set statistic collection attributes
            tqueud=tnow-atrib(1)
            call colct(tqueud,16)
            atrib(13)=1
            atrib(14)=tnow
      end if
```

Appendix A.2 Continued

```
d       write(nprnt,1776)(stadly(i),i=1,maxsta-1)
 1776   format(' station delay .sense.   ',10f5.1)
        inode = atrib(2)
c               set inode to the node that generated the packet
        atrib(10) = atrib(9) -slttim
c               set a(10) to the amount of time left after the collision
c               discrimination period
        if ( ntstus(inode) .eq. 0 ) then
c               the channel is sensed idle
d               write(nprnt,20)inode
   20           format(' tx. just occured in subr. sense ... inode = ',
     1                          i5)
                atrib(12)=tnow + waitim
c                       set a(12) to the current time plus the interframe
c                       spacing
d               write(nprnt,2525) tnow
 2525           format(' time now = ',f9.1,'    a(12) is set to tnow+waitim')
                atrib(5) = 1
c                       set a(5) to indicate a transmission event
                nodest(inode) = itrans
c                       set the node state to transmitting
                call schdl(2,waitim,atrib)
c                       schedule a transmit to be called after waiting the
c                       interframe spacing
        else
c               the channel is sensed busy
d               write(nprnt,1905) tnow
 1905           format(' time is now = ',f9.1)
d               write(nprnt,30)inode
   30           format(' defer entry packet in subr. sense ...
     1                          inode = ',i5)
                atrib(5) = 1
c                       set a(5) to a transmit event
                nodest(inode) = idefer
c                       set the node state to defering
                call filem( (nclnr-1) ,atrib)
c                       file the packet in the defer file, the packet will
c                       return to sense when the node state returns to idle
        end if
        return
        end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
      subroutine transmit
c
c short explanation:
c     The transmit event is scheduled from the sense event.  The network
c     status array is incremented, and the propagations to the nodes next
c     to the origin node are scheduled, if the origin node is node at the
c     end of the line.  A check is made to see if a collision has occured,
c     and the end of the collision discrimination period is scheduled.
c     The success event is scheduled to be called after the collision
c     discr. period.  Also the leftprop and rigtprop events are scheduled,
c     the subroutines chnlecho and collision are called.
c
      include 'params.dat'
      integer inode
      inode = atrib(2)
c             set inode to the node that generated the packet
      ntstus(inode) = ntstus(inode) + 1
c             increment the network status, so that if the node status
c             was idle it is now busy
      call chnlecho(' bgn transmit ',nodest,ntstus
     1                  ,maxsta,tnow,inode,inode)
      if ( inode .ne. 1 ) then
c             we are not at the beginning of the line and should schedule
c             a propagation 1 node to the left
              atrib(3) = inode - 1
c                     move the left propagtion marker one node to the left
              atrib(5) = 0
c                     set a(5) to indicate a propagation event
              call schdl(3,stadly(inode-1),atrib)
c                     schedule the packet to arrive at the next node
c                     to the left in the amount of time specified in
c                     the station delay array
      else
c             then we are at the beginning of the line
              atrib(3) = 0
c                     set the left propagation marker to indicate that
c                     the packet has propagated all the way to the left
      end if
```

```
      if ( inode .ne. maxsta ) then
c                we are not at the end of the line and should schedule a
c                propagation 1 node to the right
                 atrib(4) = inode + 1
c                        set the right propagation marker one node
c                        to the right
                 atrib(5) = 0
c                        set a(5) to indicate a propagation event
                 call schdl(4,stadly(inode),atrib)
c                        schedule the packet to arrive at the next node to
c                        the right in the amount of time specified in the
c                        station delay array
      else
c                we are at the end of the line
                 atrib(4) = 0
c                        set the right propagtion marker to indicate
c                        that the packet has finished propagating right
      end if
      if ( ntstus(inode) .ge. 2 ) then
c                a collision has occured during the interframe spacing
                 nodest(inode)=intrmv
c                        set the node state to indicate that a collision has
c                        occured during the interframe spacing
d                write(nprnt,2001)
 2001            format(' collision during interframe spacing')
                 call chnlecho(' coll. in discr. ',nodest,ntstus
     1                  ,maxsta,tnow,inode,inode)
                 call collision
c                        call the collision subroutine so that backoff can
c                        be determined ( amoung other things )
      else if ( nodest(inode) .eq. itrans ) then
c                the node is in the transmit state and we must schedule the
c                end of the collision discrimination period
                 atrib(5) = 1
c                        set a(5) to indicate a transmission event
d                write(nprnt,10)inode
   10            format(' enter collision discr. period ... inode = ',i5)
                 nodest(inode) = icolpr
c                        set the node state to indicate that the node is
c                        in the collision discrimination period
                 call schdl(5,slttim,atrib)
c                        schedule success to be called after the slot
c                        time ( collision discrimination period ) has
c                        elapsed, since all the other nodes will allow
c                        inode to complete it's transmission now
      else
c                if the node state was not set to transmit or the
c                ntstus was not greater than or equal to 2, then
c                this event should not have been called
d                write(nprnt,20)
   20            format(' error in node state calling event(2) transmit')
      end if
      return
      end
```

Appendix A.2 Continued

```
      subroutine leftprop
c
c short explanation:
c       Scheduled from the transmit event and itself.  This event
c       simulates the propagation of the leading edge of the packet
c       towards the left of the origin node.  Leftprop only schedules
c       itself to occur and not other events.  It calls the chnlecho
c       and collision subroutines.
c
      include 'params.dat'
      integer poslft,inode
      real sav0
      inode = atrib(2)
c             set inode to the node that generated the packet
      poslft = atrib(3)
c             set poslft to the propagate left marker
      ntstus(poslft) = ntstus(poslft) + 1
c             increment the network status of the node that the
c             leading edge of the packet has arrived to
      call chnlecho(' bgn left prop',nodest,ntstus
     1                ,maxsta,tnow,inode,poslft)
      if ((ntstus(poslft) .eq. 2).and.(nodest(poslft) .eq. icolpr)) then
c             a collision has occured at the poslft node
              sav0 = atrib(2)
c                     save the origin node
              atrib(2) = poslft
c                     set the node marker to the poslft node
              call collision
c                     call the collision subroutine
              atrib(2) = sav0
c                     return to the origin node
      end if
      atrib(3) = atrib(3) - 1
c             set the propagate left marker to the next node to the left
      poslft = atrib(3)
c             set the poslft pointer to the next node that must be
c             propagated to
      if ( poslft .ne. 0 ) then
c             we have not finished propagating to the left and must
c             schedule the packet to arrive to the next node
              atrib(5) = 0
c                     set a(5) to indicate a propagation event
              call schdl(3,stadly(poslft),atrib)
c                     schedule the packet to arrive to the next node
c                     in the amount of time specified in the stadly
c                     array, and execute the leftprop event again
      else
c             we have finished propagating to the left
              atrib(3) = 0
c                     set the left prop marker to indicated
c                     we are finished
      end if
      return
      end
```

Appendix A.2 Continued

```
      subroutine rigtprop
c
c short explanation:
c       The rigtprop event is scheduled from the transmit event and
c       itself.  It simulates the propagation of the leading edge of
c       the packet down the line towards the right of the origin node.
c       The rigtprop event only schedules itself, and calls the chnlecho
c       and collision subroutines.
c
      include 'params.dat'
      integer posrgt,inode
      real sav0
      inode = atrib(2)
c             set inode to the node that generated the packet
      posrgt = atrib(4)
c             set posrgt to the propagate left marker
      ntstus(posrgt) = ntstus(posrgt) + 1
c             increment the network status of the node that the
c             leading edge of the packet has arrived to
      call chnlecho(' bgn right prop',nodest,ntstus
     1                ,maxsta,tnow,inode,posrgt)
      if ((ntstus(posrgt) .eq. 2).and.(nodest(posrgt) .eq. icolpr)) then
c             a collision has occured at the posrgt node
              sav0 = atrib(2)
c                     save the origin node
              atrib(2) = posrgt
c                     set the node marker to the posrgt node
              call collision
c                     call the collision subroutine
              atrib(2) = sav0
c                     return to the origin node
      end if
      atrib(4) = atrib(4) + 1
c             set the propagate right marker to the next node to the right
      if ( posrgt .ne. maxsta ) then
c             we have not finished propagating to the right and must
c             schedule the packet to arrive to the next node
              atrib(5) = 0
c                     set a(5) to indicate a propagation event
              call schdl(4,stadly(posrgt),atrib)
c                     schedule the packet to arrive to the next node
c                     in the amount of time specified int the stadly
c                     array, and execute the rigtprop event again
      else
c             we have finished propagating to the right
              atrib(4) = 0
c                     set the right prop marker to indicated
c                     we are finished
      end if
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
      subroutine success
c
c short explanation:
c     The success event is scheduled from the transmit event to be
c     called after the packet has successfully made it through the
c     collision discrimination period.  In success the endtrans event
c     is scheduled to occur after the rest of the packet has been
c     transmitted, and the node state is set to terminate the packet.
c
      include 'params.dat'
      integer inode
      inode = atrib(2)
c             set inode to the node that generated the packet
      atrib(5) = 1
c             set a(5) to indicate a transmission event
d     write(nprnt,10)tnow
   10 format('  time is now        =    ',f9.1)
d     write(nprnt,20)inode
   20 format(' node ',i5,' controls the channel')
d     write(nprnt,7171) atrib(1)
 7171 format(' atrib(1)=mark time=packet creation time= ',f6.1)
      nodest(inode) = iterm
c             set the node state to terminate the packet
      call schdl(6,atrib(10),atrib)
c             schedule the endtrans event to occur once the packet has
c             finished transmitting the rest of the packet ( the amount
c             left after the collision discrimination period)
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine endtrans
c
c short explanation:
c     The endtrans event is scheduled from the success event.  The
c     network status of the origin node is decremented and the initial
c     propagation of the ending edge of the packet is scheduled to occur
c     if the origin node is node at either end of the line.  The ltfinprop
c     and rtfinprop events are scheduled, and the chnlecho and freersc
c     subroutines are called.
c
      include 'params.dat'
      integer inode
      inode = atrib(2)
c             set inode to the node that generated the packet
      ntstus(inode) = ntstus(inode) - 1
c             decrement the network status of the origin node
      call chnlecho(' end transmit ',nodest,ntstus
     1                  ,maxsta,tnow,inode,inode)
```

```
        if ( inode .ne. 1 ) then
c               we are not at the beginning of the line and should schedule
c               the ending edge to propagate 1 node to the left
                atrib(7) = inode - 1
c                       move the end of prop left marker one node
c                       to the left
                atrib(5) = 0
c                       set a(5) to indicate a propagation event
                call schdl(7,stadly(inode-1),atrib)
c                       schedule the ending edge of the packet to arrive at
c                       the next node to the left in the amount of time
c                       specified in the stadly array, and execute the
c                       ltfinprop event
        else
c               we have finished propagating the ending edge to the left
                atrib(7) = 0
c                       set the end of prop left marker to indicate we
c                       have finished
        end if
        if ( inode .ne. maxsta ) then
c               we have not finished sending the ending edge of the
c               packet to the right of the origin node and must schedule
c               the ending edge to arrive to the next node to the right
                atrib(8) = inode + 1
c                       set the end prop right marker to the next node
c                       to the right
                atrib(5) = 0
c                       set a(5) to indicate a propagation event
                call schdl(8,stadly(inode),atrib)
c                       schedule the ending edge of the packet to arrive
c                       to the next node in the amount of time specified
c                       by the stadly array, and execute the rtfinprop event
        else
c               we have finished the ending edge propagation to the right
                atrib(8) = 0
c                       set the end right prop marker to indicate we are
c                       finished
        end if
```

```
      if ( nodest(inode) .eq. iterm ) then
c                 the packet has finished transmission
                  call freersc
c                         call the freersc subroutine so that statistics can
c                         be collected and the next packet can begin the
c                         process
      else if ( nodest(inode) .eq. idefer ) then
c                 the node has a packet ready to begin the process
d                 write(nprnt,10)inode
   10             format(' node is still jamming ... inode =  ',i5)
d                 write(nprnt,15)
   15             format(' packet is defering from busy channel !!!')
      else if ( nodest(inode) .eq. ijamng ) then
c                 jam signal is over
                  nodest(inode) = iidle
c                         set the node state to idle
d                 write(nprnt,20)inode
   20             format(' node just finished jamming ... inode =  ',i5)
d                 write(nprnt,25)
   25             format(' packet will return via backoff ...
     1                    node is idle !!!')
      else
c                 the node is not in the terminate, defer, or jamming state
c                 and endtrans should not have been called
d                 write(nprnt,30)
   30             format(' error in node state calling event(6) endtrans')
      end if
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine ltfinprop
c
c short explanation:
c     The ltfinprop event is scheduled from the endtrans event and
c     itself.  In ltfinprop the ending edge of the packet traveling
c     down the line to the left of the origin node, is simulated.
c     The chnlecho and exitdefer subroutines are called.
c
      include 'params.dat'
      integer poslft,inode
      real sav0
      inode = atrib(2)
c                 set inode to the node that generated the packet
      poslft = atrib(7)
c                 set poslft to the end prop left marker
      ntstus(poslft) = ntstus(poslft) - 1
c                 decrement the network status
      call chnlecho(' end left prop',nodest,ntstus
     1                    ,maxsta,tnow,inode,poslft)
```

Appendix A.2 Continued

```
        if ((ntstus(poslft) .eq. 0).and.(nodest(poslft) .eq. idefer)) then
c                the poslft node has a packet ready to send and has sensed
c                the channel idle
                 sav0 = atrib(2)
c                        save the origin node indicator
                 atrib(2) = poslft
c                        set the node marker to the poslft node
                 call exitdefer
c                        remove the poslft node's packet from the defer file
                 atrib(2) = sav0
c                        return to the origin node
        end if
        atrib(7) = atrib(7) - 1
c                decrement the end of prop left marker to point to the
c                next node to the left
        poslft = atrib(7)
c                set poslft to the next node to be propagated to
        if ( poslft .ne. 0 ) then
c                we have not finished prop the ending edge of the packet
c                to the left and must schedule the ending edge to arrive
c                to the next node
                 atrib(5) = 0
c                        set a(5) to indicate a propagation event
                 call schdl(7,stadly(poslft),atrib)
c                        schedule the ending edge of the packet to arrive to
c                        the next node in the amount of time specified in the
c                        stadly array, and begin ltfinprop again
        else
c                we have finished propagating the ending edge
c                of the packet to
c                the left
                 atrib(7) = 0
c                        set the end of prop to the left marker to indicate
c                        we have finished
        end if
        return
        end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine rtfinprop
c
c short explanation:
c       The rtfinprop event is scheduled from the endtrans event and
c       itself.  In rtfinprop the ending edge of the packet traveling
c       down the line to the right of the origin node, is simulated.
c       The chnlecho and exitdefer subroutines are called.
c
        include 'params.dat'
        integer posrgt,inode
        real sav0
```

Appendix A.2 Continued

```
      inode = atrib(2)
c                set inode to the node that generated the packet
      posrgt = atrib(8)
c                set posrgt to the end prop right marker
      ntstus(posrgt) = ntstus(posrgt) - 1
c                decrement the network status array
      call chnlecho(' end right prop',nodest,ntstus
     1                ,maxsta,tnow,inode,posrgt)
      if ((ntstus(posrgt) .eq. 0).and.(nodest(posrgt) .eq. idefer)) then
c                the posrgt node has a packet ready to send and has
c                sensed the channel idle ( ntstus(posrgt)=0 )
                 sav0 = atrib(2)
c                        save the origin node
                 atrib(2) = posrgt
c                        set a(2) to the posrgt node
                 call exitdefer
c                        remove the posrgt node's packet from the defer file
                 atrib(2) = sav0
c                        return to the origin node
      end if
      atrib(8) = atrib(8) + 1
c                increment the end prop right marker to point to
c                the next node
      if ( posrgt .ne. maxsta ) then
c                we have not finished propagating the ending edge of the
c                packet to the right and must schedule the ending edge
c                to arrive to the next node
                 atrib(5) = 0
c                        set a(5) to indicate a prop event
                 call schdl(8,stadly(posrgt),atrib)
c                        schedule the ending edge of the packet to arrive to
c                        the next node in the amount of time specified in the
c                        stadly array, and begin rtfinprop again
      else
c                we have finished propagating the ending edge of the packet
c                to the right
                 atrib(8) = 0
c                        set the end of prop to the right marker to indicate
c                        that we have finished
      end if
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
      subroutine collision
c
c short explanation:
c     The collision subroutine is called by the transmit, leftprop,
c     after rigtprop events.  The endtrans event is scheduled to be
c     called after the jam signal is finished being sent.  The search
c     and calc_wait_backoff subroutines are called.
c
      include 'params.dat'
      real sav1,sav2
      integer inode,i
d     write(nprnt,8)
    8 format(' collsion subr. entry point ... called by lt/rt prop')
      inode = atrib(2)
c             set inode to the node that generated the packet
      sav1 = atrib(3)
c             save the prop left marker
      sav2 = atrib(4)
c             save the prop right marker
      icoll(inode) = icoll(inode) + 1
c             increment the number of collisions
d     write(nprnt,1807)(icoll(i),i=1,maxsta)
 1807 format(' # of collisions per station (coll. rout.) = ',10i3)
      if ( nodest(inode) .ne. intrmv ) then
c             the collision did not occur in the interframe spacing
              call search
c                     find the success event that was due to occur
c                     and remove it from the event calendar
      end if
      atrib(3) = sav1
c             return the prop left marker
      atrib(4) = sav2
c             return the prop right marker
      atrib(6) = atrib(6) + 1
c             increment the number of attempts for this packet
d     write(nprnt,1905) atrib(6)
 1905 format(' # of collisions for specific packet = ',f4.1)
d     write(nprnt,5555) inode
 5555 format(' node = ',i3)
      nodest(inode) = ijamng
c             set node state to jamming
      atrib(5) = 0
c             set a(5) to indicate a propagation event
      call schdl(6,rjmtim,atrib)
c             schedule the end of the jam signal
      atrib(5) = 1
c             set a(5) to indicate a transmission event
      call calc_wait_backoff
c             calculate and wait the backoff time
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Appendix A.2 Continued

```
      subroutine search
c
c short explanation:
c     The search subroutine is called by the collision subroutine.
c     In search the event that was due to occur, which would indicate
c     that the packet was successfully transmitted, is removed from
c     the event calendar, since the packet has collided.
c
      include 'params.dat'
      integer next,mmfe,nsucr,inode
      inode=atrib(2)
c             set inode to the node that generated the packet
      next = mmfe(nclnr)
c             set the pointer "next" to the first location in the event
c             calendar, which is the next event due to occur
   10 if ( next .eq. 0 ) then
c             there are not any events due to occur and the routine
c             should not have been called
d             write(nprnt,20)
   20         format(' error in collision search ...
     1                no more entries')
      return
      end if
      call copy(-next,nclnr,atrib)
c             a negitive sign in front of the rank specified tells
c             SLAM that the entry (-next) is a pointer to the location
c             rather than the specific rank, copy the attributes of the
c             entry pointed to by next into the atrib array
      if ( (atrib(2).eq.inode).and.(atrib(5).eq.1) ) then
c             this entry is a transmission event that was
c             scheduled by inode
              call rmove(-next,nclnr,atrib)
c             remove this entry from the event calendar and
c             return to the collision subroutine
      else
c             the entry copied out was not a transmission event scheduled
c             by inode, so we should continue the search
              next = nsucr(next)
c                     increment next to the following entry in the
c                     event calendar
              go to 10
c                     return to copy this entry from the event calendar
c                     and continue the process, until the event has been
c                     found
      end if
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Appendix A.2 Continued

```
      subroutine calc_wait_backoff
c
c short explanation:
c       The calc_wait_backoff subroutine is called by the collision
c       subroutine.  This subroutine checks to see if too many collisons
c       have occured and the packet should be terminated.  The backoff
c       is determined using the truncated expoential backoff algorithm.
c       In this subroutine the sense event is scheduled to occur after
c       the backoff time calculated has elapsed.
c
        include 'params.dat'
        integer iranum,inode
        real unfhi,rannum,bakoff,unfrm
        inode = atrib(2)
c               set inode to the node that generated the packet
        if ( atrib(6) .eq. mxcoll ) then
c               too many collisions have occured
d               write(nprnt,10)inode
   10           format(' too many collisions in subr. sense ...
     1                          inode = ',i5)
d               write(nprnt,1234)atrib(1)
 1234           format(' atrib(1)=mark time=packet creation time= ',
     1                          f6.1)
                excoll(inode)=excoll(inode)+1
c                       increment the number of packets lost to excessive
c                       collisions
                nodest(inode) = iterm
c                       set the node state to terminate
                return
        else
c               there has not been an excess of collisions
                if (atrib(6) .ge. 8.0) then
c                       we truncate the upper limit of the random number
                        unfhi=(2**8)-1
                else
c                       calculate the upper limit of the random number
                        unfhi=(2**atrib(6))-1
                end if
                rannum=unfrm(0.0,unfhi,iseed)
c                       determine the random number
                iranum=int(rannum+0.5)
c                       turn the random number into an integer
                bakoff=iranum*slttim
c                       determine the bakoff
        end if
d       write(nprnt,1807) bakoff
 1807   format(' collision defering transmission ... backoff = ',f10.3)
        call schdl(1,bakoff,atrib)
c               schedule the node to sense the channel when the backoff
c               time has expired
        return
        end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
      subroutine exitdefer
c
c short explanation:
c     The exitdefer subroutine is called by the ltfinprop and rtfinprop
c     events.  In this subroutine the packet that was placed in the
c     defer file, because it had sensed the channel busy, is removed.
c     Once the correct entry in the defer file is determined, that packet
c     schedules an immediate sense of the channel, i.e. the sense event
c     is scheduled.
c
      include 'params.dat'
      real sav1,sav2
      integer inode,inrank,nfind,nrank
d     write(nprnt,10)
   10 format(' exitdefer subr. entry point; called by lt/rtfin prop')
      inode = atrib(2)
c             set inode to the node that generated the packet
      sav1 = atrib(7)
c             save the end of prop left marker
      sav2 = atrib(8)
c             save the end of prop right marker
      inrank = nfind(1,( nclnr-1 ),2,0,atrib(2),0.0)
c             set inrank equal to the rank of the first entry found
c             in the defer file whose atrib(2) is exactly equal to inode
      if (inrank .eq. 0) then
c             there was not an entry in the defer file from inode and this
c             routine should not have been called
d             write(nprnt,40)
   40         format(' error in rank for exit defer ... entry
     1                not found')
      end if
      call rmove(inrank,( nclnr-1 ),atrib)
c             remove the inrank entry from the defer file and place it's
c             attributes in the atrib buffer
      atrib(5) = 1
c             set a(5) to indicate a transmission event
      call schdl(1,0.0,atrib)
c             schedule an immediate sense of the channel
      atrib(7) = sav1
c             return the end of prop left marker to a(7)
      atrib(8) = sav2
c             return the end of prop right marker to a(8)
      return
      end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Appendix A.2 Continued

```
      subroutine freersc
c
c short expanation:
c     The freersc subroutine is called by the endtrans event when a
c     packet has experienced excessive collisions or has been
c     successfully transmitted.  In freersc one unit of resource of the
c     origin node is released, so that the next packet waiting in the
c     node queue can begin the process.  In addition much of the
c     statistics collection is done in freersc.
c
      include 'params.dat'
      integer inode,i,badpak
      real tsys,taccss,tchnl,totsys
      badpak=0
c             assume a successful packet
d     write(nprnt,10)
   10 format(' freersc subr. entry point; called by success,sense')
d     write(nprnt,1776) atrib(2),atrib(6)
 1776 format(' origin = node ',f3.1,' # of coll. atrib(6) = ',f4.1)
      inode = atrib(2)
c             set inode to the node that generated the packet
      call free(inode,1)
c             free 1 unit of resource inode [ note:  the node number
c             equals the resource number ]
      icnt(inode) = icnt(inode) + 1
c             increment the number of transmitted packets counter
d     write(nprnt,1905)(icnt(i),i=1,maxsta)
 1905 format(' packet count per station (in freersc) = ',10i3)
d     write(nprnt,2001)(icoll(i),i=1,maxsta)
 2001 format(' # of collisions per station (freersc routine) = ',10i3)
      if ( atrib(6) .eq. mxcoll ) then
c             the packet had experienced excessive collisions
              badpak=1
c                     set badpak to indicate an unsuccessful packet
      end if
      if ( badpak .eq. 1 ) then
c             the packet was unsuccessful
              tgood=0.0
c                     do not include the time to transmit in
c                     the throughput
              bitsbd=bitsbd + atrib(11)
c                     add the number of bits in this packet to the total
c                     number of unsuccessful bits
      else
c             the packet was successfully transmitted
              tgood=tnow-atrib(12)
c                     calculated the amount of time required to
c                     transmit the packet
              bitsgd=bitsgd + atrib(11)
c                     add the number of bits in this packet to the total
c                     number of successfully transmitted bits
      end if
```

Appendix A.2 Continued

```
d       write(nprnt,1807) tgood
 1807   format(' time sending good packet = ',f8.2)
        timegd(inode)=timegd(inode)+tgood
c               add the time to send the packet to the total time spent
c               sending packets successfully from the specific node
        attempts( atrib(6)+1 ) = attempts( atrib(6) + 1 ) + 1
c               increment the number of packets successful in the specific
c               number of attempts required by this particular packet
        tsys = tnow - atrib(1)
c               calculate the total system delay
d       write(nprnt,4545) tsys
 4545   format(' time in system (tsys) = ',f8.1)
        call colct(tsys,inode)
c               collect statistics on the total system delay for this
c               node
        if ( atrib(17) .eq. 1.0 ) then
c               the packet was successful on its first attempt to access
c               the network
                frstat(inode)=frstat(inode)+1
c                       increment the number of packets successfully
c                       transmitted from inode on there first attempt
c                       to access the network
        end if
        taccss=atrib(12)-atrib(14)
c               calculate the amount of time required to access the network
c               for this particular packet
        call colct(taccss,17)
c               collect statistics on the amount of time to access the net
        if (  badpak .eq. 0 ) then
c               the packet was successfully transmitted
                tchnl=tnow-atrib(12)
c                       calculate the delay through the channel
                call colct(tchnl,18)
c                       collect statistics on the delay through the channel
        end if
        totsys=tnow-atrib(1)
c               calculate the total delay
        call colct(totsys,19)
c               collect statistics on the total delay through the system
d       write(nprnt,1957) inode
 1957   format(' leaving freersc subr.  inode = ',i3)
        return
        end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Appendix A.2 Continued

```
      subroutine chnlecho(inhedr,in1arr,in2arr,maxx,time,numsta,numpos)
c
c short explanation:
c       Prints the header specified as the event or subroutine which
c       called chnlecho, the current time, origin node, the network
c       status array and the node state array.
c
      integer i,maxx,numsta,numpos
      integer in1arr(*),in2arr(*)
      real time
      character inhedr*15
d     write(6,15)
   15 format('-------------------------------------------------------'
     1 ,'--------------------------')
d     write(6,20)inhedr,time,numsta,numpos
   20 format(a15,'  time now =',f9.1,'   origin node =',i4,'
     1 local node =',i4)
d     write(6,3030)(in1arr(i),i=1,maxx)
 3030 format('         node state:   ',7i3)
d     write(6,4040)(in2arr(i),i=1,maxx)
 4040 format('         net status:   ',7i3)
d     write(6,15)
      return
      end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Appendix A.3 Listing of the INTLC Subroutine

```
      subroutine intlc
c
c     The intlc subroutine has been included in the discrete event model
c     to initialize some SLAM variables and user defined variables.
c
      include 'params.dat'
      integer i
c
c initialize user variables:
      do  i=1,maxsta
            excoll(i) = 0
            frstat(i) = 0
            icnt(i)   = 0
            icoll(i)  = 0
            nodest(i) = 5
            ntstus(i) = 0
            timegd(i) = 0.0
      end do
c
      do  i=1,mxcoll
            attempts(i)=0
      end do
c
      bitsgd=0
      bitsbd=0
c
c initialize the station delay array ( the user can put any value for
c delay between nodes, i.e. the nodes do not have to be equally spaced)
c
      do i=1,maxsta-1
            stadly(i)=2.75/(maxsta-1)
      end do
c
c
c
      return
      end
```

Appendix A.4 Listing of the OTPUT Subroutine

```
      subroutine otput
c
c short explanation:
c       The otput subroutine has been included in the discrete event model
c       so that the simulation results of some specific performance
c       indicators cold be reported.  These specific things include
c       throughput per node, overall network throughput, number of
c       collisions which occured per node and total, collisions per
c       millisecond, number of packets from each node and total number that
c       were successful on thier first attempt to access the net, number of
c       packets successful after a given number of attempts, the total
c       number of packets transmitted from each station and the total
c       number transmitted, the number of packets discarded due to excessive
c       collisions, the number of bits successfully transmitted, and
c       the number of bits unsuccessfully transmitted.
c
      include 'params.dat'
c
      integer i,totpak,totfrt,totdcd,totcoll
      real tottim,stathu,totthu,frtper,ftaper,disper,colpms,peratt
c
      totcoll=0
      tottim=0.0
      totfrt=0
      totdcd=0
      totpak=0
c
      do i=1,maxsta
            totpak=totpak+icnt(i)
      end do
c
c skip to the next page
      write(6,1905)
 1905 format('1    ')
c
   10 format('    ')
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  page 1  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
      do i=1,maxsta
            stathu=timegd(i)/xx(25)
            write(6,1827) i,timegd(i),stathu
 1827       format(' time spent sending good packets from station '
     1            ,i3,' = ',f15.2,'  throughput = ',f5.3)
            tottim=tottim+timegd(i)
      end do
      write(6,10)
      write(6,10)
```

Appendix A.4 Continued

```
c
c
         totthu=tottim/xx(25)
         write(6,9898) tottim,totthu
 9898    format(' the total time spent with good packets = ',f12.2,
     1           '   total throughput = ',f5.3)
         write(6,10)
         write(6,10)
c
         do i=1,maxsta
                 write(6,60) i,icoll(i)
   60            format(' total # of collisions that occured at station '
     1                   ,i5,' = ',i8)
                 totcoll=totcoll+icoll(i)
         end do
         write(6,10)
         write(6,10)
c
         colpms=1000*float(totcoll)/xx(25)
         write(6,1000) totcoll,colpms
 1000    format(' total # of collisions = ',i8,
     1           '  collisions per millisecond = ',f9.4)
         write(6,1905)
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  page 2  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
         do i=1,maxsta
                 write(6,1957) i,frstat(i)
 1957            format(' # of packets successful on the first'
     1                   ,' attempt to access net from ',i3,' = ',i5)
                 totfrt=totfrt+frstat(i)
         end do
         write(6,10)
         frtper=100*(float(totfrt)/float(totpak))
         write(6,8585) totfrt,frtper
 8585    format('          total # of 1st access = ',i9,
     1           '       % of 1st access = ',f8.4)
         write(6,10)
         write(6,10)
c
```

```
      do i=1,mxcoll
            if ( i .eq. 1 ) then
                  peratt=100*float(attempts(i))/float(totpak)
                  write(6,1010) i,attempts(i),peratt
 1010             format(' # of packets successful on the ',i2,
     1                  'st  attempt = ',i8,
     2                  '     percentage = ',f6.3)
            else if ( i .eq. 2 ) then
                  peratt=100*float(attempts(i))/float(totpak)
                  write(6,2020) i,attempts(i),peratt
 2020             format(' # of packets successful on the ',i2,
     1                  'nd  attempt = ',i8,
     2                  '     percentage = ',f6.3)
            else if ( i .eq. 3 ) then
                  peratt=100*float(attempts(i))/float(totpak)
                  write(6,3030) i,attempts(i),peratt
 3030             format(' # of packets successful on the ',i2
     1                  ,'rd  attempt = ',i8,
     2                  '     percentage = ',f6.3)
            else
                  peratt=100*float(attempts(i))/float(totpak)
                  write(6,1111) i,attempts(i),peratt
 1111             format(' # of packets successful on the ',i2
     1                  ,'th  attempt = ',i8,
     2                  '     percentage = ',f6.3)
            end if
      end do
      write(6,10)
c
      ftaper=100*(float(attempts(1))/float(totpak))
      write(6,2233) ftaper
 2233 format('              % successful on 1st attempt = ',f8.4)
c
      write(6,1905)
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<  page 3  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
      do i=1,maxsta
            write(6,30) i,icnt(i)
   30       format(' total # of packets transmitted from station '
     1            ,i5,' = ',i8)
      end do
      write(6,10)
c
      write(6,1807) totpak
 1807 format(' total # of packets transmitted = ',i13)
      write(6,10)
      write(6,10)
```

Appendix A.4 Continued

```
c
        do i=1,maxsta
                write(6,7777) i,excoll(i)
 7777           format(' # of discarded packets from ',i3,' = ',i5)
                totdcd=totdcd+excoll(i)
        end do
        write(6,10)
        disper=100*(float(totdcd)/float(totpak))
        write(6,9696) totdcd,disper
 9696   format('     total # of discarded packets = ',i9,
     1          '     % of packets discarded = ',f8.4)
        write(6,10)
        write(6,10)
c
        write(6,2222) bitsgd
 2222   format(' total # of successful bits transmitted = ',i15)
        write(6,10)
c
        write(6,3333) bitsbd
 3333   format(' total # of unsuccessful (discarded) bits = ',i8)
c
        return
        end
```

Appendix A.5 Listing of the PARAMS File

```
      implicit none
c
c define user parameters:
c
      integer maxsta,mxcoll,idefer,itrans,icolpr,ijamng,iterm
     1       ,iidle,intrmv,iseed
c
      real    waitim,rjmtim,slttim
c
      parameter ( maxsta = 5 )
      parameter ( mxcoll = 16 )
      parameter ( waitim = 9.6 )
      parameter ( rjmtim = 10.88435374 )
      parameter ( idefer = 0 )
      parameter ( itrans = 1 )
      parameter ( icolpr = 2 )
      parameter ( ijamng = 3 )
      parameter ( iterm  = 4 )
      parameter ( iidle  = 5 )
      parameter ( intrmv = 6 )
      parameter ( slttim = 5.5 )
c
c define slam random number stream for backoff selection
      parameter ( iseed = 5 )
c
c define slam variables:
      integer ii,mfa,mstop,nclnr,ncrdr,nprnt,nnrun,nnset,ntape
      real    atrib,dd,ddl,dtnow,ss,ssl,tnext
     1       ,tnow,xx
c
      common/scom1/ atrib(100),dd(100),ddl(100),dtnow,ii,mfa,mstop,nclnr
     1       ,ncrdr,nprnt,nnrun,nnset,ntape,ss(100),ssl(100)
     2       ,tnext,tnow,xx(100)
c
c define user variables:
      integer ntstus,nodest,icnt,icoll,frstat,excoll,attempts
     1       ,bitsgd,bitsbd
      real    stadly,timegd,tgood
c
      common/iucom/ ntstus(maxsta),nodest(maxsta),icnt(maxsta)
     1       ,icoll(maxsta),stadly(maxsta-1),excoll(maxsta)
     2       ,frstat(maxsta),timegd(maxsta),tgood,attempts(mxcoll)
     3       ,bitsgd,bitsbd
```

APPENDIX B LISTING OF THE MODIFIED CSMA/CD SIMULATION MODEL

Appendix B.1 Listing of the Modified Network Model

```
GEN,ED FRIEDMAN,MULTIRATE LOAD CNTRL,8/20/84,1,NO,NO;
LIMITS,22,18,750;
;
;
; Describe attributes:
;
;       atrib(1) = packet creation time
;       atrib(2) = node identification
;       atrib(3) = propagation left marker, if left prop is finished
;                  then atrib(3)=0
;       atrib(4) = propagation right marker, if right prop is finished
;                  then atrib(4)=0
;       atrib(5) = if atrib(5)=0 then event being scheduled is propagation,
;                  if atrib(5)=1 then event being scheduled is transmission
;       atrib(6) = counter for the number of attempts to transmit
;       atrib(7) = end of propagation left marker, if atrib(7)=0 then
;                  finished prop left
;       atrib(8) = end of propagation right marker, if atrib(8)=0 then
;                  finished prop right
;       atrib(9) = packet length in microseconds
;       atrib(10)= atrib(9)-slttim = amount of time left after the
;                  collision discrimination period
;       atrib(11)= number of bits per packet
;       atrib(12)= marks the time transmission is attempted, gets reset
;                  each time a retransmission is attempted
;       atrib(13)= while packet is in the AWAIT NODE atrib(13)=0,
;                  set to 1 when packet leaves the queue, used to collect
;                  statistics on the amount of time a packet is queued,
;                  tqueud= atrib(14) minus atrib(1)if atrib(13)=0
;       atrib(14)= marks the time when a packet leaves the node queue (the
;                  AWAIT), used to collect statistics on the time required
;                  to access the network, taccss=atrib(12)-atrib(14)
;       atrib(15)= set to 1 if the voice packet lifetime is exceeded
;       atrib(16)= set to 1 if packet originated at a data node, set to
;                   2 if packet originated at a voice node
;       atrib(17)= first access attribute, if the packet is trying to
;                  transmit for the first time then atrib(17) is set
;                  to 1 from 0, on subsequent attempts to transmit
;                  atrib(17) is incremented
;       atrib(18)= the generation period for the particular voice packet
```

Appendix B.1 Continued

```
; Describe SLAM Global Variables:
;
;       xx(1) = the next generation period to be used by a voice node
;       xx(2) = the next generation period to be used by a voice node
;       xx(3) = not used ( node 3 is a data node )
;       xx(4) = the next generation period to be used by a voice node
;       xx(5) = not used ( node 5 is a data node )
;       xx(6) = the next generation period to be used by a voice node
;       xx(7) = the next generation period to be used by a voice node
;       xx(8) = the next generation period to be used by a voice node
;       xx(9) = not used ( node 9 is a data node )
;       xx(10)= the next generation period to be used by a voice node
;       xx(11)= the next generation period to be used by a voice node
;       xx(12)= the next generation period to be used by a voice node
;       xx(13)= the next generation period to be used by a voice node
;       xx(14)= not used ( node 14 is a data node )
;       xx(15)= the next generation period to be used by a voice node
;       xx(16)= the next generation period to be used by a voice node
;       xx(17)= not used ( node 17 is a data node )
;       xx(18)= the next generation period to be used by a voice node
;       xx(19)= the next generation period to be used by a voice node
;       xx(20)= the next generation period to be used by a voice node
;
;       xx(21)-xx(24) = not used
;
;       xx(25) = the simulated time ( ttfin )
;
;       xx(26)-xx(29) = not used
;
;       xx(30) = the mean interarrival rate for the data nodes
;
;
; Files Used by SLAM:
;
;       files  1-20  reserved for the AWAIT NODE at each station
;       file    21   channel QUEUE NODE
;       file    22   defer file
;       file    23   event calendar
;
;
;
STAT,1,NODE  1 SYS TIME
STAT,2,NODE  2 SYS TIME
STAT,3,NODE  3 SYS TIME
STAT,4,NODE  4 SYS TIME
STAT,5,NODE  5 SYS TIME
STAT,6,NODE  6 SYS TIME
STAT,7,NODE  7 SYS TIME
STAT,8,NODE  8 SYS TIME
STAT,9,NODE  9 SYS TIME
STAT,10,NODE 10 SYS TIME
```

Appendix B.1 Continued

```
STAT,11,NODE 11 SYS TIME
STAT,12,NODE 12 SYS TIME
STAT,13,NODE 13 SYS TIME
STAT,14,NODE 14 SYS TIME
STAT,15,NODE 15 SYS TIME
STAT,16,NODE 16 SYS TIME
STAT,17,NODE 17 SYS TIME
STAT,18,NODE 18 SYS TIME
STAT,19,NODE 19 SYS TIME
STAT,20,NODE 20 SYS TIME
;
STAT,21,QUEUE DLY   DATA
STAT,22,ACCSS DLY   DATA
STAT,23,CHNL DELAY  DATA
STAT,24,TOTSYS DLY  DATA
STAT,25,QUEUE DLY  VOICE
STAT,26,ACCSS DLY  VOICE
STAT,27,CHNL DLY   VOICE
STAT,28,TOTSYS DLY VOICE
;
;
NETWORK;
;
;
      RESOURCE/PACK1,1/PACK2,2/PACK3,3/PACK4,4/PACK5,5/PACK6,6/
               PACK7,7/PACK8,8/PACK9,9/PACK10,10/PACK11,11/
               PACK12,12/PACK13,13/PACK14,14/PACK15,15/PACK16,16/
               PACK17,17/PACK18,18/PACK19,19/PACK20,20;
;
;
;
;
;     MODEL OF THE NUMBER OF STATIONS ON THE ETHER
;     ===========================================
;
;-------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
      CREATE,XX(1),2900,1;        create packets at node 1
      ASSIGN,ATRIB(2)=1,
             ATRIB(9)=768.0,
             ATRIB(11)=768.0,
             ATRIB(15)=0.0,
             ATRIB(16)=2,
             ATRIB(17)=0,
             ATRIB(18)=XX(1),1;   set attributes
      AWAIT(1),PACK1;             packets wait for previous packet to finish
      ACT,,,CHNL;                 immediate branch to the channel
;
;-------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
       CREATE, XX(2), 7300, 1;
       ASSIGN, ATRIB(2)=2.0,
               ATRIB(9)=768.0,
               ATRIB(11)=768.0,
               ATRIB(15)=0.0,
               ATRIB(16)=2,
               ATRIB(17)=0,
               ATRIB(18)=XX(2), 1;
       AWAIT(2), PACK2;
       ACT, , , CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*data*>>>>>>>>>>
       CREATE, EXPON(XX(30)), 2000, 1;
       ASSIGN, ATRIB(2)=3.0,
               ATRIB(9)=2048,
               ATRIB(11)=2048,
               ATRIB(15)=0.0,
               ATRIB(16)=1,
               ATRIB(17)=0, 1;
       AWAIT(3), PACK3;
       ACT, , , CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
       CREATE, XX(4), 14000, 1;
       ASSIGN, ATRIB(2)=4.0,
               ATRIB(9)=768.0,
               ATRIB(11)=768.0,
               ATRIB(15)=0.0,
               ATRIB(16)=2,
               ATRIB(17)=0,
               ATRIB(18)=XX(4), 1;
       AWAIT(4), PACK4;
       ACT, , , CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*data*>>>>>>>>>>
        CREATE,EXPON(XX(30)),100000,1;
        ASSIGN,ATRIB(2)=5,
                ATRIB(9)=2048,
                ATRIB(11)=2048,
                ATRIB(15)=0,
                ATRIB(16)=1,
                ATRIB(17)=0,1;
        AWAIT(5),PACK5;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(6),2600,1;
        ASSIGN,ATRIB(2)=6,
                ATRIB(9)=768.0,
                ATRIB(11)=768.0,
                ATRIB(15)=0.0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(6),1;
        AWAIT(6),PACK6;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(7),8600,1;
        ASSIGN,ATRIB(2)=7,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0.0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(7),1;
        AWAIT(7),PACK7;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE, XX(8), 6700, 1;
        ASSIGN, ATRIB(2)=8,
                ATRIB(9)=768.0,
                ATRIB(11)=768.0,
                ATRIB(15)=0.0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(8), 1;
        AWAIT(8), PACK8;
        ACT, , , CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*data*>>>>>>>>>>
        CREATE, EXPON(XX(30)), 5000, 1;
        ASSIGN, ATRIB(2)=9.0,
                ATRIB(9)=2048,
                ATRIB(11)=2048,
                ATRIB(15)=0.0,
                ATRIB(16)=1,
                ATRIB(17)=0, 1;
        AWAIT(9), PACK9;
        ACT, , , CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE, XX(10), 2, 1;
        ASSIGN, ATRIB(2)=10.0,
                ATRIB(9)=768.0,
                ATRIB(11)=768.0,
                ATRIB(15)=0.0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(10), 1;
        AWAIT(10), PACK10;
        ACT, , , CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(11),390,1;
        ASSIGN,ATRIB(2)=11.0,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(11),1;
        AWAIT(11),PACK11;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(12),2000,1;
        ASSIGN,ATRIB(2)=12,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(12),1;
        AWAIT(12),PACK12;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(13),11000,1;
        ASSIGN,ATRIB(2)=13,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(13),1;
        AWAIT(13),PACK13;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*data*>>>>>>>>>>
        CREATE,EXPON(XX(30)),70000,1;
        ASSIGN,ATRIB(2)=14,
                ATRIB(9)=2048,
                ATRIB(11)=2048,
                ATRIB(15)=0,
                ATRIB(16)=1,
                ATRIB(17)=0,1;
        AWAIT(14),PACK14;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(15),6000,1;
        ASSIGN,ATRIB(2)=15,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(15),1;
        AWAIT(15),PACK15;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
        CREATE,XX(16),4200,1;
        ASSIGN,ATRIB(2)=16,
                ATRIB(9)=768,
                ATRIB(11)=768,
                ATRIB(15)=0,
                ATRIB(16)=2,
                ATRIB(17)=0,
                ATRIB(18)=XX(16),1;
        AWAIT(16),PACK16;
        ACT,,,CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*data*>>>>>>>>>>
        CREATE, EXPON(XX(30)), 40000, 1;
        ASSIGN, ATRIB(2)=17.0,
                ATRIB(9)=2048,
                ATRIB(11)=2048,
                ATRIB(15)=0.0,
                ATRIB(16)=1,
                ATRIB(17)=0, 1;
        AWAIT(17), PACK17;
        ACT,,, CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
;       CREATE, XX(18), 5400, 1;
;       ASSIGN, ATRIB(2)=18,
;               ATRIB(9)=768,
;               ATRIB(11)=768,
;               ATRIB(15)=0,
;               ATRIB(16)=2,
;               ATRIB(17)=0,
;               ATRIB(18)=XX(18), 1;
;       AWAIT(18), PACK18;
;       ACT,,, CHNL;
;
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
;       CREATE, XX(19), 3400, 1;
;       ASSIGN, ATRIB(2)=19,
;               ATRIB(9)=768,
;               ATRIB(11)=768,
;               ATRIB(15)=0,
;               ATRIB(16)=2,
;               ATRIB(17)=0,
;               ATRIB(18)=XX(19), 1;
;       AWAIT(19), PACK19;
;       ACT,,, CHNL;
;
;-------------------------------------------------------------------------
```

Appendix B.1 Continued

```
;-------------------------------------------------------------------------
;
;<<<<<<<<<<*voice*>>>>>>>>>>
;       CREATE,XX(20),8600,1;
;       ASSIGN,ATRIB(2)=20,
;              ATRIB(9)=768,
;              ATRIB(11)=768,
;              ATRIB(15)=0,
;              ATRIB(16)=2,
;              ATRIB(17)=0,
;              ATRIB(18)=XX(20),1;
;       AWAIT(20),PACK20;
;       ACT,,,CHNL;
;
;-------------------------------------------------------------------------
;
;
;       COMBINE STATIONS TO FORM CHANNEL
;       ================================
;
CHNL   QUEUE(21);                   dump entities into queue
;
       ACT;                         immediate branch to channel model
;
       EVENT,1;                     gateway to discrete event model, maps
;                                   entities into event 1 ( sense )
;
;
       ASSIGN,XX(25)=60000000;      set xx(25) equal to ttfin
       END;
INIT,0,60000000;
FIN;
```

Appendix B.2 Listing of the Modified and New Events and Subroutines

```
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine event(i)
c
c short explanation:
c       The event subroutine is required by SLAM in all discrete event
c       models, the user will schedule an event to occur and the event
c       subroutine calls the particular event when the time scheduled
c       is reached.
c
      go to (1,2,3,4,5,6,7,8,9,10),i
    1 call sense
      return
    2 call transmit
      return
    3 call leftprop
      return
    4 call rigtprop
      return
    5 call success
      return
    6 call endtrans
      return
    7 call ltfinprop
      return
    8 call rtfinprop
      return
    9 call detrate
      return
   10 call loadchg
      return
      end
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine sense
c
c short explanation:
c       The sense event is scheduled from the calc_wait_backoff and
c       exitdefer subroutines.  In sense statistic collection is done,
c       and statistic collection attributes are set.  Then sense looks
c       at the network status array to see if the channel is idle, if
c       the channel is idle a transmit event is scheduled, if the channel
c       is not idle the packet gets put in the defer file.
c
      include 'params.dat'
      integer inode,i
      real tqueud
      inode = atrib(2)
c               set inode to the node that generated the packet
```

Appendix B.2 Continued

```
      if ( atrib(17) .eq. 0.0  .and.  atrib(16) .eq. 2 ) then
c               this is the first time this voice packet has tried to
c               access the network
                call detgen
c                       determine the generation period
                xx(inode) = atrib(18)
c                       set the generation period for the next packet
      end if
      atrib(17)=atrib(17)+1
c               increment the first access atrib
      if  ( atrib(13) .eq. 0 ) then
c                       the packet has just left the node queue, so collect
c                       statistics and set statistic collection attributes
                tqueud=tnow-atrib(1)
                if ( atrib(16) .eq. 1 ) then
c                       the current packet is data
                        call colct(tqueud,21)
                else
c                       the current packet is voice
                        call colct(tqueud,25)
                end if
                atrib(13)=1
                atrib(14)=tnow
      end if
d     write(nprnt,1776)(stadly(i),i=1,maxsta-1)
 1776 format(' station delay .sense.  ',10f5.1)
      atrib(10) = atrib(9) -slttim
c               set a(10) to the amount of time left after the collision
c               discrimination period
      if ( tnow-atrib(1) .ge. atrib(18)  .and.  atrib(16) .eq. 2 ) then
c               the current packet is voice and it's lifetime has
c               been exceeded
                atrib(15) = 1.0
c                       set atrib(15) to indicate that the lifetime
c                       is exceeded
                nodest(inode) = iterm
c                       set the node state to terminate
                lstpak(inode)=lstpak(inode)+1
c                       collect lost packet statistics
                call freersc
c                       call the freersc subroutine so that statistics
c                       can be collected and the next packet can begin
c                       the process
                return
      end if
```

Appendix B.2 Continued

```
        if ( ntstus(inode) .eq. 0 ) then
c                   the channel is sensed idle
d                   write(nprnt,20)inode
   20               format(' tx. just occured in subr. sense ... inode = ',
     1                                i5)
                    atrib(12)=tnow + waitim
c                           set a(12) to the current time plus the interframe
c                           spacing
d                   write(nprnt,2525) tnow
 2525               format(' time now = ',f9.1,'    a(12) is set to tnow+waitim')
                    atrib(5) = 1
c                           set a(5) to indicate a transmission event
                    nodest(inode) = itrans
c                           set the node state to transmitting
                    call schdl(2,waitim,atrib)
c                           schedule a transmit to be called after waiting the
c                           interframe spacing
        else
c                   the channel is sensed busy
d                   write(nprnt,1905) tnow
 1905               format(' time is now = ',f9.1)
d                   write(nprnt,30)inode
   30               format(' defer entry packet in subr. sense ...
     1                                inode = ',i5)
                    atrib(5) = 1
c                           set a(5) to a transmit event
                    nodest(inode) = idefer
c                           set the node state to defering
                    call filem( (nclnr-1) ,atrib)
c                           file the packet in the defer file, the packet will
c                           return to sense when the node state returns to idle
        end if
        return
        end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine detgen
c
c short explanation:
c       The detgen subroutine is called from the sense event, for each
c       new arriving voice packet.  The voice coding rate is determined
c       by truncating the calculated ratenow, and the 18th attribute is
c       set to the generation period.  The counters are incremented.
c
        include 'params.dat'
c
        if ( atrib(2) .eq. 8 ) then
c                   the packet came from the 8th node and the current
c                   time and voice coding rate are written
           write(nd8x2,1008) tnow/1000,1000*atrib(9)*capcty/atrib(18)
 1008      format(1x,f13.3,'      ',f7.1)
        end if
```

Appendix B.2 Continued

```
c
c truncate ratenow to one of the 4 chosen values of coding rate:
        if ( ratenow .le. 28000 ) then
                rate = choice(1)
        else if ( ratenow .lt. 36000 ) then
                rate = choice(2)
        else if ( ratenow .lt. 44000 ) then
                rate = choice(3)
        else
                rate = choice(4)
        end if
c
        atrib(18) = 1000000*atrib(9)*capcty/rate
c               calculate the generation period from the rate
c               that was chosen
c
        if ( atrib(16) .eq. 2 ) then
c               the packet being processed is voice
                if ( atrib(2) .eq. 8 ) then
c                  write out information on the single node (node 8)
c                  increment counters
                        write(nd8,5008) tnow/1000,rate/1000
                        write(nd8x2,5008) tnow/1000,rate/1000
 5008                   format(1x,f13.3,'      ',f7.1)
                        num8=num8+1
                        sum8=sum8+rate/1000
                        num8oa=num8oa+1
                        sum8oa=sum8oa+rate/1000
                end if
        end if
c
c
        return
        end
```

Appendix B.2 Continued

```
c++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine detrate
c
c short explanation:
c       The detrate event is scheduled from the intlc subroutine and
c       itself.  A new value for ratenow is calculated using the feedback
c       equation.  The new value for ratenow is truncated to one of the
c       chosen values.     The collision counter 'colcnt' is set to zero.
c
        include 'params.dat'
        real bigq,litq,ravg
c
c set the parameters used in the feedback equation:
        bigq = 3.3
        litq = 13000
        ravg = 33000
c
        call schdl(9,period,atrib)
c               schedule detrate to be executed after period
c               microseconds have
c               elapsed
        write(out2,1776) tnow/1000,colpms
 1776   format(1x,f13.3,'      ',f8.4)
        colpms=1000*float(colcnt)/period
c               calculate the collisions per millisecond
        sumc=sumc+colpms
        numc=numc+1
        write(out2,1807) tnow/1000,colpms
 1807   format(1x,f13.3,'      ',f8.4)
        colcnt=0
c               set the collision counter to zero
c
c
        write(out1,1905) tnow/1000,rateout/1000
 1905   format(1x,f13.3,'      ',f7.1)
        sumrate=sumrate+rateout/1000
        numrate=numrate+1
        sumoa=sumoa+rateout/1000
        numoa=numoa+1
c
        ratenow = ravg + litq*(bigq-colpms)
c               calculate the new value or ratenow using the feedback
c               equation
c
```

Appendix B.2 Continued

```
c truncate ratenow:
        if ( ratenow .le. 28000 ) then
                rateout = choice(1)
        else if ( ratenow .lt. 36000 ) then
                rateout = choice(2)
        else if ( ratenow .lt. 44000 ) then
                rateout = choice(3)
        else
                rateout = choice(4)
        end if
c
        write(out1,1492) tnow/1000,rateout/1000
 1492   format(1x,f13.3,'      ',f7.1)
c
        return
        end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine loadchg
c
c short explanation:
c       The loadchg event is scheduled from the intlc subroutine and
c       itself.  The amount of data traffic is changed dynamically
c       during the simulation when this event is executed.
c
        include 'params.dat'
        integer i,j,iran,ldtst
        real    ran,unfrm,ulo,uhi
        i=int(tnow/loadtim)
c               calculated 'i' it gives an indication of how long the
c               simulation has been going
c
        if ( allrand .eq. 'yes' ) then
c               the load is chosen randomly using the random load array
                xx(30) = rload(i+1)
        else
                the loads are non-random
                xx(30) = load(i)
        end if
c
c calculate statistics:
        datadel(i) = sumdel/numdel
        avgcpm(i)  = sumc/numc
        loadu(i+1) = xx(30)
        avgrate(i) = sumrate/numrate
        avg8(i)    = sum8/num8
c
c
```

Appendix B.2 Continued

```
        if ( tnow .ne. loadtim*float(numchg-1) ) then
c               the simulation has more than 1 'loadtim' seconds left
c               to go, so schedule this to be executed in 'loadtim'
c               microseconds
                call schdl(10,loadtim,atrib)
        end if
c
c set counters:
        sumc    = 0.0
        numc    = 0.0
        sumrate = 0.0
        numrate = 0.0
        sum8    = 0.0
        num8    = 0.0
        sumdel  = 0.0
        numdel  = 0.0
c
        return
        end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine collision
c
c short explanation:
c       The collision subroutine is called by the transmit, leftprop,
c       and rigtprop events.  The endtrans event is scheduled to be
c       called after the jam signal is finished being sent.  The search
c       and calc_wait_backoff subroutines are called.
c
        include 'params.dat'
        real sav1,sav2
        integer inode,i
d       write(nprnt,8)
    8   format(' collsion subr. entry point ... called by lt/rt prop')
        inode = atrib(2)
c               set inode to the node that generated the packet
        sav1 = atrib(3)
c               save the prop left marker
        sav2 = atrib(4)
c               save the prop right marker
        icoll(inode) = icoll(inode) + 1
c               increment the number of collisions
        colcnt = colcnt + 1
c               increment the number of collisions for periodic
c               calculation of collisions per millisecond
d       write(nprnt,1807)(icoll(i),i=1,maxsta)
 1807   format(' # of collisions per station (coll. rout.) = ',10i3)
        if ( nodest(inode) .ne. intrmv ) then
c               the collision did not occur in the interframe spacing
                call search
c                       find the success event that was due to occur
c                       and remove it from the event calendar
        end if
```

```
        atrib(3) = sav1
c               return the prop left marker
        atrib(4) = sav2
c               return the prop right marker
        atrib(6) = atrib(6) + 1
c               increment the number of attempts for this packet
d       write(nprnt,1905) atrib(6)
 1905   format(' # of collisions for specific packet = ',f4.1)
d       write(nprnt,5555) inode
 5555   format(' node = ',i3)
        nodest(inode) = ijamng
c               set node state to jamming
        atrib(5) = 0
c               set a(5) to indicate a propagation event
        call schdl(6,rjmtim,atrib)
c               schedule the end of the jam signal
        atrib(5) = 1
c               set a(5) to indicate a transmission event
        call calc_wait_backoff
c               calculate and wait the backoff time
        return
        end
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        subroutine calc_wait_backoff
c
c short explanation:
c       The calc_wait_backoff subroutine is called by the collision
c       subroutine.  The backoff is determined using the truncated
c       expoential backoff algorithm.  In this subroutine the sense
c       event is scheduled to occur after the backoff time calculated
c       has elapsed.  There is also a check to see if too many collisions
c       have occured.
c
        include 'params.dat'
        integer iranum,inode
        real unfhi,rannum,bakoff,unfrm,trund,trunv
        inode = atrib(2)
c               set inode to the node that generated the packet
        trund=10
c               set the truncation for data packets
        trunv=9
c               set the truncation for voice
```

Appendix B.2 Continued

```
      if ( atrib(6) .eq. mxcoll ) then
c                 too many collisions have occured
d                 write(nprnt,10)inode
   10             format(' too many collisions ...
     1                          inode = ',i5)
d                 write(nprnt,1234)atrib(1)
 1234             format(' atrib(1)=mark time=packet creation time= ',
     1                          f6.1)
                  excoll(inode)=excoll(inode)+1
c                         increment the number of packets lost to
c                         excessive collisions
                  nodest(inode) = iterm
c                         set the node state to terminate
                  return
      else
c                 there has not been an excess of collisions
                  if ( atrib(16) .eq. 1 ) then
                          if (atrib(6) .ge. trund) then
                                  unfhi=(2**trund)-1
                          else
                                  unfhi=(2**atrib(6))-1
                          end if
                          rannum=unfrm(0.0,unfhi,iseed)
                          iranum=rannum
                          bakoff=iranum*slttim
                  else
                          if (atrib(6) .ge. trunv) then
                                  unfhi=(2**trunv)-1
                          else
                                  unfhi=(2**atrib(6))-1
                          end if
                          rannum=unfrm(0.0,unfhi,iseed)
                          iranum=rannum
                          bakoff=iranum*slttim
                  end if
      end if
d     write(nprnt,1807) bakoff
 1807 format(' collision defering transmission ... backoff = ',f10.3)
      call schdl(1,bakoff,atrib)
c                 schedule the node to sense the channel when the backoff
c                 time has expired
      return
      end
```

```
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      subroutine freersc
c
c short expanation:
c     The freersc subroutine is called by the endtrans event when a
c     packet has experienced excessive collisions or has been
c     successfully transmitted.  In freersc one unit of resource of
c     the origin node is released, so that the next packet waiting
c     in the node queue can begin the process.  In addition much of
c     the statistics collection is done in freersc.
c
      include 'params.dat'
      integer inode,i,badpak
      real tsys,taccss,tchnl,totsys
      badpak=0
c             assume a successful packet
d     write(nprnt,10)
   10 format(' freersc subr. entry point; called by success,sense')
d     write(nprnt,1776) atrib(2),atrib(6)
 1776 format(' origin = node ',f3.1,' # of coll. atrib(6) = ',f4.1)
      inode = atrib(2)
c             set inode to the node that generated the packet
      call free(inode,1)
c             free 1 unit of resource inode [ note:  the node number
c             equals the resource number ]
      icnt(inode) = icnt(inode) + 1
c             increment the number of transmitted packets counter
d     write(nprnt,1905)(icnt(i),i=1,maxsta)
 1905 format(' packet count per station (in freersc) = ',10i3)
d     write(nprnt,2001)(icoll(i),i=1,maxsta)
 2001 format(' # of collisions per station (freersc routine) = ',10i3)
      if ( atrib(6) .eq. mxcoll ) then
c             the packet had experienced excessive collisions
              badpak=1
c                     set badpak to indicate an unsuccessful packet
      end if
      if ( atrib(15) .eq. 1 ) then
c             the packet is voice and it's lifetime has been exceeded
              badpak = 1
c                     set badpak to indicate an unsuccessful packet
      end if
      if ( atrib(15) .eq. 1 ) then
c             the packet was discarded due to an excess of lifetime
              rwlst(inode)=rwlst(inode)+1
c                     increment the number of packets lost in a row
              if ( rwlst(inode) .gt. rwlstp(inode) ) then
c                     set rwlstp to indicate larger value
                      rwlstp(inode)=rwlst(inode)
              end if
      else
c             packet was not lost to an excess of lifetime
              rwlst(inode)=0
c                     set the rwlst to restart the row counting
      end if
```

```
        if ( atrib(6) .eq. mxcoll ) then
c               the packet was discarded due to excessive collisions
                rwdisc(inode)=rwdisc(inode)+1
c                       increment the number of packets lost in a row
                if ( rwdisc(inode) .gt. rwdiscp(inode) ) then
c                       set rwdiscp to indicate larger value
                        rwdiscp(inode)=rwdisc(inode)
                end if
        else
c               the packet was not lost to an excess of collisions
                rwdisc(inode)=0
c                       set the rwdisc to restart the row counting
        end if
        if ( badpak .eq. 1 ) then
c               the packet was unsuccessful
                tgood=0.0
c                       do not include the time to transmit in
c                       the throughput
                bitsbd=bitsbd + atrib(11)
c                       add the number of bits in this packet to the
c                       total number of unsuccessful bits
        else
c               the packet was successfully transmitted
                tgood=tnow-atrib(12)
c                       calculated the amount of time required to
c                       transmit the packet
                bitsgd=bitsgd + atrib(11)
c                       add the number of bits in this packet to the total
c                       number of successfully transmitted bits
        end if
d       write(nprnt,1807) tgood
 1807   format(' time sending good packet = ',f8.2)
        timegd(inode)=timegd(inode)+tgood
c               add the time to send the packet to the total time spent
c               sending packets successfully from the specific node
        attempts( atrib(6)+1 ) = attempts( atrib(6) + 1 ) + 1
c               increment the number of packets successful in the specific
c               number of attempts required by this particular packet
        tsys = tnow - atrib(1)
c               calculate the total system delay
d       write(nprnt,4545) tsys
 4545   format(' time in system (tsys) = ',f8.1)
        call colct(tsys,inode)
c               collect statistics on the total system delay for this
c               node
        if ( atrib(17) .eq. 1.0 ) then
c               the packet was successful on its first attempt to access
c               the network
                frstat(inode)=frstat(inode)+1
c                       increment the number of packets successfully
c                       transmitted from inode on there first attempt
c                       to access the network
        end if
```

Appendix B.2 Continued

```
        taccss=atrib(12)-atrib(14)
c               calculate the amount of time required to access the
c               network for this particular packet
        if ( atrib(16) .eq. 1 ) then
c               the packet is data
                call colct(taccss,22)
c                       collect access statistics for data packets
        else
c               the packet is voice
                call colct(taccss,26)
c                       collect access statistics for voice packets
        end if
        if (  badpak .eq. 0 ) then
c               the packet was successfully transmitted
                tchnl=tnow-atrib(12)
c                       calculate the delay through the channel
                if ( atrib(16) .eq. 1 ) then
c                       the packet is data
                        call colct(tchnl,23)
c                               collect channel statistics for
c                               data packets
                else
c                       the packet is voice
                        call colct(tchnl,27)
c                               collect channel statistics for
c                               voice packets
                end if
        end if
        totsys=tnow-atrib(1)
c               calculate the total delay
        if ( atrib(16) .eq. 1 ) then
c               the packet is data
                call colct(totsys,24)
c                       collect system delay statistics for data packets
        else
c               the packet is voice
                call colct(totsys,28)
c                       collect systerm delay statistics for voice packets
        end if
        if ( atrib(16) .eq. 1 ) then
c               the packet is data and the statistic collection variables,
c               used to report the data delay during the time interval that
c               a particular load was present, must be updated
                sumdel=sumdel+totsys
                numdel=numdel+1
        end if
d       write(nprnt,1957) inode
 1957   format(' leaving freersc subr.  inode = ',i3)
        return
        end
c*************************************************************************
```

Appendix B.3 Listing of the Modified INTLC Subroutine

```
      subroutine intlc
c
c     The intlc subroutine has been included in the discrete event
c     model to initialize some SLAM variables and user defined
c     variables.
c
      include 'params.dat'
      integer i
c
c initialize user variables:
c
      do  i=1,maxsta
            excoll(i) = 0
            frstat(i) = 0
            icnt(i)   = 0
            icoll(i)  = 0
            lstpak(i) = 0
            nodest(i) = 5
            ntstus(i) = 0
            rwlst(i)  = 0
            rwlstp(i) = 0
            rwdisc(i) = 0
            rwdiscp(i)= 0
            timegd(i) = 0.0
      end do
c
      bitsbd=0
      bitsgd=0
c
      do  i=1,mxcoll
            attempts(i)=0
      end do
c
c initialize the station delay array ( the user can put any value for
c delay between nodes, i.e. the nodes do not have to be equally spaced)
      do i=1,maxsta-1
            stadly(i)=4.5/(maxsta-1)
      end do
c
c
c initalize attribute 18 and the SLAM global variables:
      atrib(18)= 16000
c
      do i = 1,maxsta
            xx(i) = 16000.0
      end do
c
c
c set the possible coding rates:
      choice(1) = 24000
      choice(2) = 32000
      choice(3) = 40000
      choice(4) = 48000
```

Appendix B.3 Continued

```
c
c initalize the counters:
        colcnt = 0
        colpms = 0
        numrate= 0
        sumrate= 0
        numoa  = 0
        sumoa  = 0
        num8   = 0
        sum8   = 0
        num8oa = 0
        sum8oa = 0
        sumc   = 0
        numc   = 0
        sumdel = 0
        numdel = 0
c set the period over which the collisions per millisecond will
c be calculated:
        period = 32000
c
c initalize the rate ( first packets coding rate ):
        rate   = 48000
        ratenow= 48000
        rateout= 48000
c
c set the load array ( used if non-random loads are desired ):
        do i=1,numlds
                load(i) = 500*2048/(10*float(i))
        end do
c
c set the random load array ( contains the possible loads to be used ):
        rload(1) = 500*2048/15
        rload(2) = 500*2048/5
        rload(3) = 500*2048/20
        rload(4) = 500*2048/15
        rload(5) = 500*2048/5
        rload(6) = 500*2048/10
        rload(7) = 500*2048/25
        rload(8) = 500*2048/5
        rload(9) = 500*2048/15
        rload(10)= 500*2048/20
        rload(11)= 500*2048/25
        rload(12)= 500*2048/20
c
c initialize the data load:
        xx(30)   = 500*2048/15
        loadu(1) = 15
```

Appendix B.3 Continued

```
c
c if the user wants the load to be determined randomly for the simulation
c then allrand should be set to 'yes', otherwise set it to 'no'
      allrand  = 'yes'
c
c set the time between load changes:
      loadtim = 5000000
c
c set up files for storing the time/rate data for all the nodes (tr.dat),
c the time/collisions per millisecond (ct.dat), the time/rate data for
c node 8 (tr8.dat), a file of time/rate data from node 8 that is to be
c plotted (tr8plot.dat):
      call opnchk(out1,'[friedman.csmacd.mrvoi.data]tr.dat'
     1                          ,'new','for')
      call opnchk(out2,'[friedman.csmacd.mrvoi.data]ct.dat'
     1                          ,'new','for')
      call opnchk(nd8,'[friedman.csmacd.mrvoi.data]tr8.dat'
     1                          ,'new','for')
      call opnchk(nd8x2,'[friedman.csmacd.mrvoi.data]tr8plot.dat'
     1                          ,'new','for')
c
c write initial values to the files:
      write(out1,1807) tnow/1000,rate/1000
 1807 format(1x,f13.3,'      ',f7.1)
      write(out2,1776) tnow/1000,colpms
 1776 format(1x,f13.3,'      ',f8.4)
      write(nd8x2,2001) tnow/1000,rate/1000
 2001 format(1x,f13.3,'      ',f7.1)
c
c
c schedule the collisions per millisecond to be determined,
c and the next load:
      call schdl(9,period,atrib)
      call schdl(10,loadtim,atrib)
c
      return
      end
```

Appendix B.4 Listing of the Modified OTPUT Subroutine

```
      subroutine otput
c
c short explanation:
c       The otput subroutine has been included in the discrete event
c       model so that the simulation results of some specific performance
c       indicators could be reported.  These specific things include
c       throughput per node, overall network throughput, number of
c       collisions which occured per node and total, collisions per
c       millisecond, number of packets from each node and total number
c       that were successful on their first attempt to access the net,
c       number of packets successful after a given number of attempts,
c       the total number of packets transmitted from each station and
c       the total number transmitted, the number of packets discarded
c       due to excessive collisions, the number of bits successfully
c       transmitted, and  the number of bits unsuccessfully transmitted.
c       In addition the number and percentage of packets lost to an
c       excess of packet lifetime per node and total, the number of
c       packets lost in a row per node, the total number and percentage
c       of lost voice packets (includes those lost due to excess of
c       collisions and those lost to an excess of packet lifetime),
c       Also, the average coding rate overall and node 8, the percentage
c       of the load that was data, the average collisions per millisecond,
c       and the data delay are reported for the time range that a
c        particular data load was present.  As well as the overall average
c       coding rate for all nodes, the overall average coding rate for
c       node 8, the overall average percentage of the load that was data,
c       and the overall average rate of collisions per millisecond.
c
      include 'params.dat'
c
      integer i,totpak,totfrt,totdcd,totcoll,totlst,totvdcd
      real  tottim,stathu,totthu,frtper,ftaper,disper,totcpms,peratt
     1          ,perlst,totper,vperlst,time1,time2,sumload,avgload
     2          ,sumcpm,avgcpmall,sumdd,perdisc
c
      sumload = 0
      sumcpm  = 0
      totvdcd = 0
      totlst  = 0
      totcoll = 0
      tottim  = 0.0
      totfrt  = 0
      totdcd  = 0
      totpak  = 0
c
      do i=1,maxsta
              totpak=totpak+icnt(i)
      end do
c
c skip to the next page
      write(6,1905)
 1905 format('1    ')
c
   10 format('    ')
```

```
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   page 1   >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
        do i=1,maxsta
                stathu=timegd(i)/xx(25)
                write(6,1827) i,timegd(i),stathu
 1827           format(' time spent sending good packets from station '
     1                  ,i3,' = ',f15.2,'  throughput = ',f5.3)
                tottim=tottim+timegd(i)
        end do
        write(6,10)
        write(6,10)
c
c
        totthu=tottim/xx(25)
        write(6,9898) tottim,totthu
 9898   format(' the total time spent with good packets = ',f12.2,
     1          '    total throughput = ',f5.3)
        write(6,10)
        write(6,10)
c
        do i=1,maxsta
                write(6,60) i,icoll(i)
   60           format(' total # of collisions that occured at station '
     1                  ,i5,' = ',i8)
                totcoll=totcoll+icoll(i)
        end do
        write(6,10)
        write(6,10)
c
        totcpms=1000*float(totcoll)/xx(25)
        write(6,1000) totcoll,totcpms
 1000   format(' total # of collisions = ',i8,
     1          '  total average collisions per millisecond = ',f9.4)
        write(6,1905)
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   page 2   >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
        do i=1,maxsta
                write(6,1957) i,frstat(i)
 1957           format(' # of packets successful on the first'
     1                  ,' attempt to access net from ',i3,' = ',i5)
                totfrt=totfrt+frstat(i)
        end do
        write(6,10)
        frtper=100*(float(totfrt)/float(totpak))
        write(6,8585) totfrt,frtper
 8585   format('         total # of 1st access = ',i9,
     1          '        % of 1st access = ',f8.4)
        write(6,10)
        write(6,10)
```

Appendix B.4 Continued

```
c
        do i=1,mxcoll
                if ( i .eq. 1 ) then
                        peratt=100*float(attempts(i))/float(totpak)
                        write(6,1010) i,attempts(i),peratt
 1010                   format(' # of packets successful on the ',i2,
     1                          'st  attempt = ',i8,
     2                          '     percentage = ',f6.3)
                else if ( i .eq. 2 ) then
                        peratt=100*float(attempts(i))/float(totpak)
                        write(6,2020) i,attempts(i),peratt
 2020                   format(' # of packets successful on the ',i2,
     1                          'nd  attempt = ',i8,
     2                          '     percentage = ',f6.3)
                else if ( i .eq. 3 ) then
                        peratt=100*float(attempts(i))/float(totpak)
                        write(6,3030) i,attempts(i),peratt
 3030                   format(' # of packets successful on the ',i2
     1                         ,'rd  attempt = ',i8,
     2                          '     percentage = ',f6.3)
                else
                        peratt=100*float(attempts(i))/float(totpak)
                        write(6,1111) i,attempts(i),peratt
 1111                   format(' # of packets successful on the ',i2
     1                         ,'th  attempt = ',i8,
     2                          '     percentage = ',f6.3)
                end if
        end do
        write(6,10)
c
        ftaper=100*(float(attempts(1))/float(totpak))
        write(6,2233) ftaper
 2233   format('              % successful on 1st attempt = ',f8.4)
c
        write(6,1905)
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<   page 3   >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
        do i=1,maxsta
                write(6,30) i,icnt(i)
   30           format(' total # of packets transmitted from station '
     1                 ,i5,' = ',i8)
        end do
        write(6,10)
c
        write(6,1807) totpak
 1807   format(' total # of packets transmitted = ',i13)
        write(6,10)
        write(6,10)
```

Appendix B.4 Continued

```
c
        write(6,1551)
 1551   format(' ---packets discarded due to excessive collisions---')
        do i=1,maxsta
                perdisc=100*float(excoll(i))/float(icnt(i))
                write(6,3773) i,excoll(i),perdisc,rwdiscp(i)
 3773           format(1x,'station ',i2,':  # of packets discarded = '
     1                 ,i3,3x,'percentage = ',f5.2,3x
     2                 ,'discarded in a row = ',i2)
                if ( i.ne.3 .and. i.ne.5 .and. i.ne.9 .and. i.ne.14
     1                          .and. i.ne.17 ) then
                        totvdcd = totvdcd + excoll(i)
                end if
                totdcd=totdcd+excoll(i)
        end do
        write(6,10)
        disper=100*(float(totdcd)/float(totpak))
        write(6,9696) totdcd,disper
 9696   format('     total # of discarded packets = ',i9,
     1          '     % of packets discarded = ',f8.4)
        write(6,10)
        write(6,10)
c
        write(6,2222) bitsgd
 2222   format(' total # of successful bits transmitted = ',i15)
        write(6,10)
c
        write(6,3333) bitsbd
 3333   format(' total # of unsuccessful (discarded) bits = ',i8)
c
        write(6,1905)
c
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<   page 4   >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
        write(6,1990)
 1990   format(' ---packets discarded due to excess of lifetime---')
        do i=1,maxsta
                if ( lstpak(i) .eq. 0  .or.  icnt(i) .eq. 0 ) then
                        perlst=0.0
                else
                        totlst = totlst + lstpak(i)
                        perlst = 100*float(lstpak(i))/float(icnt(i))
                end if
                write(6,2000) i,lstpak(i),perlst,rwlstp(i)
 2000           format(' station ',i3,':  # of packets lost = '
     1                 ,i4,'     percentage = ',f5.2,
     2                 '          # of packets lost in a row = '
     3                 ,i2)
        end do
```

Appendix B.4 Continued

```
        totper = 100*float(totlst)/float(totpak)
        write(6,10)
        write(6,3000) totper,totlst
 3000   format(' average percentage of lost packets = ',f7.4,
     1          '       total # of packets lost = ',i6)
c
c
        write(6,10)
        write(6,10)
        vperlst = 100*float(totlst+totvdcd)/float(totpak)
        write(6,2990) totlst+totvdcd,vperlst
 2990   format(1x,'----- total of lost voice packets -----',/,5x
     1          ,'number = ',i9,5x,'percentage = ',f7.4)
c
c
        write(6,1905)
c
c<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   page 5   >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
c
c
        avg8(numchg)    = sum8/num8
        avgrate(numchg) = sumrate/numrate
        avgcpm(numchg)  = sumc/numc
        datadel(numchg) = sumdel/numdel
        write(6,3990)
 3990   format(1x,'average rate',26x,'average',3x,'average',/
     1          ,1x,'total',2x,'node 8',2x,'time range',4x,'% load'
     2          ,3x,'colpms',3x,'data delay, ms')
        do i=1,numchg
                sumdd = sumdd + datadel(i)/1000
                sumcpm=sumcpm+avgcpm(i)
                time1 = loadtim*float(i-1)/1000000
                if ( i.ne.numchg ) then
                        time2 = loadtim*float(i)/1000000
                else
                        time2 = xx(25)/1000000
                end if
                if ( i.ne.1 ) then
                        loadu(i) = 500*2048/loadu(i)
                end if
                sumload = sumload + loadu(i)
```

Appendix B.4 Continued

```
                if ( time2.lt.10 ) then
                  write(6,1001) avgrate(i),avg8(i),time1,time2,loadu(i)
     1                         ,avgcpm(i),datadel(i)/1000
 1001               format(2x,f4.1,3x,f4.1,3x,f4.1,'-',f3.1,7x,f4.1
     1                         ,3x,f7.4,4x,f8.4)
                else if ( time2 .lt.  100 ) then
                  write(6,1002) avgrate(i),avg8(i),time1,time2,loadu(i)
     1                         ,avgcpm(i),datadel(i)/1000
 1002               format(2x,f4.1,3x,f4.1,3x,f4.1,'-',f4.1,6x,f4.1
     1                         ,3x,f7.4,4x,f8.4)
                else
                  write(6,1003) avgrate(i),avg8(i),time1,time2,loadu(i)
     1                         ,avgcpm(i),datadel(i)/1000
 1003               format(2x,f4.1,2x,f5.1,3x,f4.1,'-',f5.1,5x,f4.1
     1                         ,3x,f7.4,4x,f8.4)
                end if
          end do
          write(6,10)
          avgload=sumload/float(numchg)
          avgcpmall=sumcpm/float(numchg)
          write(6,3992) sumoa/numoa,sum8oa/num8oa,avgload,avgcpmall
     1                    ,sumdd/float(numchg)
 3992     format(1x,'            overall average rate = ',f7.4,/,1x
     1            ,'overall average rate for node 8 = ',f7.4,/,1x
     2            ,'            average percent load = ',f7.4,/,1x
     3            ,'          overall average colpms = ',f7.4,/,1x
     4            ,'       average data delay in ms =',f8.4)
c
c
          call uclose(out1)
          call uclose(out2)
          call uclose(nd8)
          call uclose(nd8x2)
c
c
          return
          end
```

Appendix B.5 Listing of the PARAMS File

```
      implicit none
c
c define user parameters:
c
      integer icolpr,idefer,iidle,ijamng,intrmv,iterm,itrans,iseed
     1       ,maxsta,mxcoll,numchg,numlds,numrlds
c
      real    capcty,genper,rjmtim,slttim,waitim
c
      parameter ( idefer = 0 )
      parameter ( itrans = 1 )
      parameter ( icolpr = 2 )
      parameter ( ijamng = 3 )
      parameter ( iterm  = 4 )
      parameter ( iidle  = 5 )
      parameter ( intrmv = 6 )
c
      parameter ( capcty = 1.0 )
      parameter ( maxsta = 17  )
      parameter ( mxcoll = 16  )
      parameter ( numchg = 12  )
      parameter ( numlds = 4   )
      parameter ( numrlds= 12  )
      parameter ( rjmtim = 4.8 )
      parameter ( slttim = 9.0 )
      parameter ( waitim = 9.6 )
c
c
c define slam random number stream for backoff selection
      parameter ( iseed = 5 )
c
c
c define SLAM variables:
c
      integer ii,mfa,mstop,nclnr,ncrdr,nprnt,nnrun,nnset,ntape
      real    atrib,dd,ddl,dtnow,ss,ssl,tnext
     1       ,tnow,xx
c
      common/scom1/ atrib(100),dd(100),ddl(100),dtnow,ii,mfa,mstop,nclnr
     1       ,ncrdr,nprnt,nnrun,nnset,ntape,ss(100),ssl(100)
     2       ,tnext,tnow,xx(100)
c
c
c define user variables:
c
      integer ntstus,nodest,icnt,icoll,frstat,excoll,attempts
     1       ,bitsgd,bitsbd,lstpak,rwlst,rwlstp,colcnt
     2       ,out1,out2,nd8,nd8x2,rwdisc,rwdiscp
      real    stadly,timegd,tgood,choice,rateout
     1       ,period,colpms,load,rload,rate,ratenow
     2       ,sumrate,numrate,avgrate,sum8,num8,avg8
     3       ,sumoa,numoa,sum8oa,num8oa,loadtim,loadu
     4       ,sumc,numc,avgcpm,datadel,sumdel,numdel
```

Appendix B.5 Continued

```
      character*3 allrand
c
      common/iucom/ ntstus(maxsta),nodest(maxsta),icnt(maxsta)
     1          ,icoll(maxsta),stadly(maxsta-1),excoll(maxsta)
     2          ,frstat(maxsta),timegd(maxsta),tgood,attempts(mxcoll)
     3          ,bitsgd,bitsbd,lstpak(maxsta),rwlst(maxsta)
     4          ,rwlstp(maxsta),choice(4),out1,out2
     5          ,colcnt,period,colpms,load(numlds),rload(numrlds)
     6          ,rate,ratenow,rateout
     7          ,nd8,nd8x2,sumrate,numrate,avgrate(numchg)
     8          ,sum8,num8,avg8(numchg),sumoa,numoa,sum8oa,num8oa
     9          ,loadtim,loadu(numchg),sumc,numc,avgcpm(numchg)
     1          ,allrand,datadel(numchg),sumdel,numdel
     2          ,rwdisc(maxsta),rwdiscp(maxsta)
```