TACTICAL AIR CONTROL SYSTEM
SIMULATION PROGRAM

THESIS

AFIT/GCS/EE/82D-24  Donald P. McCanless
                      Captain  USAF

Approved for public release; distribution unlimited.
TACTICAL AIR CONTROL SYSTEM
SIMULATION PROGRAM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Donald P. McCanless, B. S.
Captain USAF
Graduate Computer Systems
December 1982

Approved for public release; distribution unlimited.
Acknowledgements

I would like to acknowledge the assistance provided by my thesis advisor, Lieutenant Colonel Harold Carter. Without his patience and guidance this effort would not have been completed. I would also like to thank my readers, Major Walter Seward and Dr. David Barr, for finding the time in their busy schedules to read this document and provide suggestions for improvement. Lastly, I would like to thank Professor Dan Reynolds for all of the enthusiastic support he provided.
Contents

Acknowledgements .......................................................... ii
List of Figures .............................................................. iv
Abstract ........................................................................... v
I. Introduction ...................................................................... 1
II. The Pre-Processor ............................................................ 2
   Introduction ....................................................................... 2
   JLAMPP Capabilities ......................................................... 2
      Nodes ........................................................................... 2
      Terminate Nodes .......................................................... 3
      Message Creation ......................................................... 3
      Routing ......................................................................... 3
      Feasibility Study ......................................................... 3
      Selected Capabilities ...................................................... 3
      Design Stages ................................................................ 4
      Program Evaluation ....................................................... 4
      Completed Design ......................................................... 5
III. Program Installation ........................................................ 9
     Introduction ..................................................................... 9
     Array Size Selection ...................................................... 9
     Compilation .................................................................... 9
     Enlarging SLAM ............................................................. 10
IV. SLAMPP Validation ........................................................ 11
    Introduction ..................................................................... 11
    Test Case ......................................................................... 11
    Test Case Validation ...................................................... 11
    Error Detection ............................................................... 12
V. Summary ........................................................................... 13
Bibliography ........................................................................ 14
Appendix A: Program Listing ................................................ 15
Appendix B: Operating Instructions ........................................ 51
Appendix C: Input Errors ....................................................... 62
Appendix D: Interpreting the SLAM Output ............................ 66
Appendix E: Test and Evaluation ............................................ 70
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Program Flow</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>SLAM Main Routine</td>
<td>10</td>
</tr>
</tbody>
</table>
Abstract

The Tactical Air Force's Interoperability Group wants to be able to simulate communication networks. The simulation language for alternative modeling (SLAM) was proposed as the basis for the simulation. However, this language requires too much effort to make any changes to the network to be simulated. A Pre-Processor was written in FORTRAN 77 which allows the user to describe the network in a convenient manner, and produces the necessary SLAM source code to perform the simulation.
I. Introduction

The Tactical Air Force Interoperability Group (TAFIG) located at Langley Air Force Base, Virginia, wishes to be able to simulate the flow of message traffic through the Tactical Air Control System. This simulation will allow TAFIG to determine the number of messages going across particular communication channels and statistics about the queues which may form at nodes (Morrison, 1981). Current models, concluded one study, are too broad and it was recommended that a specific set of problems be defined and selected model development be continued (Bennett, 1970: ix).

A simulation language for alternative modeling (SLAM) was selected as the basis for the simulation. However, every change to the communication network being modeled would require rewriting the SLAM source code. This constraint was not seen as being "user-friendly" enough for a production system.

In order to make the ease of input acceptable, a SLAM Pre-Processor was proposed. This Pre-Processor would take, as inputs, a description of the network to be simulated along with descriptions of the message traffic and produce, as output, the necessary SLAM source code to model the network. This SLAM source code would, in turn, be input into the SLAM program for the actual simulation.
II. The Pre-Processor

Introduction

In order to determine the feasibility of such a system as the SLAM Pre-Processor (SLAMPP), a subset of the desired capabilities was selected and implemented. Following this evaluation, the system was expanded to include all of the desired capabilities.

SLAMPP Capabilities

The SLAMPP was designed to allow the user to simulate communication networks. To the SLAMPP, a network consists of message creations and queues (processing locations), along with the path a message will follow through the various queues.

In order to describe a communication network, SLAMPP must allow the user to describe the network to be simulated. The items which may be described are listed below.

Nodes. A node, or queue, is a location at which a message will receive some type of handling. The maximum number of messages which may be waiting at a node at any given time can be specified. The name of an alternate node may (optionally) be supplied to which a message will "balk" should the node's capacity be exceeded, or "blocking" may be specified which causes messages to "back up" in the system when capacity is exceeded. If neither balking nor blocking is specified, the message will simply be lost (without any indication to the user) if capacity has been reached.

Two time durations are associated with a node. The first allows
the user to specify how long it takes to process a message at this node, while the second specifies how long transmission to the next node will take. If either of these two times are not specified, a default time of zero will be assumed.

Terminate Nodes. Terminate nodes are locations at which a message leaves the network.

Message Creation. The rate at which a message of a particular type is created can be specified. Additionally, the starting time for this message and the number of messages to be created are also specified.

Routing. Each message may be routed through the nodes following paths which the user supplies. Also, an alternate balk route can be supplied should a message be redirected to a balk node.

Feasibility Study

In order to determine the feasibility of the SLAMPF, a subset of the capabilities was selected and implemented.

Selected Capabilities. Only the "bare" essential capabilities were selected for the feasibility study. Additionally, these capabilities were (in some cases) simplified.

Node definition was included; however, balk nodes were not implemented. Terminate nodes were included exactly as they would be included in the final version, as was message creation. Path definition did not support balk paths, nor blocking. Message processing times and transmit times between nodes were ignored.

Design Stages. Following the selection of the capabilities to be studied, the feasibility fell into several stages. First, a logical and easy-to-follow method was developed to input the network description to
the Pre-Processor. These input formats, with some modifications and additions, remained in the final version.

Next, a small network was designed and, using the input formats mentioned previously, a SLAMPP description of this network was coded.

A great deal of information was necessary to build even this simple implementation. Storing and manipulating this data quickly became confusing. In order to try and clarify this problem, the variables and data structures to store this information was designed. Although some modifications occurred during the course of the project, the basic design remained the same.

Now that the exact capabilities for the feasibility study were known, and the inputs and data structures were designed, flowcharts were produced which showed the steps necessary to implement the project. The flowcharts began at a very "high-level" and were iteratively refined until the detail necessary to design the program was obtained.

Using these flowcharts, the FORTRAN source code was written. Although some logic errors did exist, they proved to be relatively minor and easy to correct.

**Program Evaluation.** After the SLAMPP was written and debugged, the Pre-Processor output was examined for correctness. This examination included a visual inspection of the output file to verify the SLAM code produced by the Pre-Processor was as expected, and an execution of this code by the SLAM program to ensure the code would indeed be accepted by the SLAM system.

This examination showed that the SLAMPP output was valid and acceptable to the SLAM program. Additionally, it showed that the basic
concept behind the Pre-Processor was viable and continued development was warranted.

Completed Design

Following the feasibility study, the remaining capabilities were incorporated into the final SLAMPP. Due to modularity of design these additions were easily installed into the framework developed earlier (see Appendix A for a listing of the SLAMPP program).

The basic flow of the program is shown in Figure 1, while Appendix B contains detailed operating instructions. First, all of the nodes are defined, followed by all of the terminate nodes. After these definitions comes the start of the first message. Each message consists of one message definition, followed by an optional process and transmit time definition, followed by the name of the next node along the path. Successive steps in the path are indicated by repetition of process and transmit times and node name. Additional messages are input by placing another message definition into the data deck and repeating the above steps.

After all messages have been described, information about balk routes (if any are present) is input. This process is similar to message definition, however, the message definition card is replaced by a balk route definition card. The process and transmit time and path definitions are the same as above. One balk definition is included for each message that can possibly balk.

The validity of the inputs is verified prior to building the SLAM source code (a summary of the input errors which can be detected is
contained in Appendix C). If all inputs are valid, the SLAM source is constructed.

The SLAM source built by the Pre-Processor now serves as input to the SLAM program for the actual simulation. The outputs produced by the SLAM program are described in Appendix D.
Figure 1. Program Flow (Part One)
Figure 1. Program Flow (Part Two)
III. Program Installation

Introduction

Prior to execution of the SLAM Pre-Processor, the program must be compiled. To minimize the amount of computer memory required, certain changes can be made to the program to decrease array sizes.

Array Size Selection

Due to the way SLAMPP is written, the user has the flexibility to vary the maximum number of messages (NOMSG) the Pre-Processor can handle, the maximum number of terminate nodes (NOTRM), and the maximum number of nodes (NONOD).

SLAMPP is written entirely in FORTRAN 77. To allow for variable dimension sizes for arrays, each subroutine and function contains a PARAMETER statement (Katzan, 1978: 77) which defines each of the three variables sizes. Prior to compilation, the user should decide on appropriate values for these parameters and change each PARAMETER statement accordingly.

Since each of the three parameters are used to determine array sizes, the larger the parameter sizes, the larger the amount of computer memory required to execute SLAMPP. Therefore, care should be used to select values which are both realistic and conservative.

Compilation

As mentioned previously, SLAMPP is written in FORTRAN 77. After any changes to the PARAMETER statements are made, the Pre-Processor is
ready to be compiled and saved using whatever installation dependent procedures are required.

**Enlarging SLAM**

Figure 2 shows the main driver for the SLAM program. Depending upon the size of the SLAMPP network to be simulated, certain changes may be necessary to the SLAM main routine to accommodate the network. Although these changes are part of the SLAM program itself, and not part of the Pre-Processor, they are listed here for the user's convenience.

Three items must be changed: the dimension sizes of NSET and QSET, and the value for NNSET. Each of these three values must be the same to ensure a successful SLAM run. Typically, the default value is 5,000.

```c
PROGRAM MAINRT (INPUT, OUTPUT, TAPE7, TAPE5=INPUT, &
                  TAPE6=OUTPUT)
C
C THIS ROUTINE IS INCLUDED SO THAT THE ARRAY SIZE
C FOR NNSET/QSET CAN BE CHANGED TO ACCOMODATE
C LARGE SIMULATION RUNS.
C
DIMENSION NSET(7000)
COMMON /SCOMI/ ATRIB(100), DD(100), DDL(100), &
                  DTHOW, II, MFA, MSTOP, NCLNR, NCRDR, NPRNT, &
                  TNEXT, TNOW, XX(100)
COMMON QSFT(7000)
EQUIVALENCE (NSET(1), QSET(1))
NNSET = 7000
NCRDR = 5
NPRNT = 6
NTAPE = 7
CALL SLAM
STOP
END
```

Figure 2. SLAM Main Driver
IV. SLAMPP Validation

Introduction

In order to check the validity of the SLAMPP (that is, verify that SLAMPP produces a SLAM network which represents the intention of the inputs), a test case was designed and evaluated. In addition to this large test case, over 30 smaller networks were input to the Pre-Processor during development to assure a valid Pre-Processor was in fact produced. Also, tests were performed to ensure that invalid inputs would be detected.

Test Case

The test case used for validation was designed to exercise every capability of SLAMPP. Specifically, nodes of all three types were used (some with balking, some with blocking, and some with neither). Nine different messages were included to demonstrate each of the nine valid distribution types (see Appendix B). Process and transit times included at least one example of each of the nine distributions also, plus examples of the zero default time which will be generated if not specified otherwise. Since balking was specified on two nodes, balk route descriptions were required in addition to regular route descriptions. See Appendix E for a listing of the SLAMPP inputs and the associated SLAM source which was generated.

Test Case Validation

After the test network was designed, the SLAMPP description of the
network was written and input into the Pre-Processor. The SLAM source output was then carefully examined to ensure that a valid network representation had been produced. Also, the Pre-Processor output was executed through the SLAM program to verify it would execute successfully. Both the examination and the execution showed the SLAMPP output to contain no known discrepancies.

Error Detection

Given valid inputs, SLAMPP will produce a corresponding SLAM network; however, it is equally important that the Pre-Processor recognize invalid inputs to prevent the creation of a bad network.

Many types of input validations are performed by the Pre-Processor. The errors which may be detected are summarized in Appendix C. To be certain that each input validation point worked correctly, test cases were devised which intentionally (and unintentionally) exercised each point. Since the validity of any SLAM network created using invalid input data is questionable, no SLAM source is generated if any errors are detected.
V. Summary

The Tactical Air Forces Interoperability Group needs to be able to simulate the flow of message traffic through the Tactical Air Control System. The simulation language for alternative modeling (SLAM) was used as the basis for the simulation. A Pre-Processor was constructed to permit a user-friendly operation.

The Pre-Processor was written in FORTRAN 77, and produced the source code to be used by the SLAM program. The Pre-Processor output was validated and found to be correct.
Bibliography


Appendix A: Program Listing

Following is the FORTRAN 77 listing of the source code for the SLAM Pre-Processor.
PROGRAM SLAMPP
C 
C THIS PROGRAM ACCEPTS (FROM FILE 10) INPUT CARDS DESCRIBING THE 
C NETWORK TO BE SIMULATED, CHECKS THEIR VALIDITY, AND (IF NO 
C ERRORS ARE DETECTED) PRODUCES THE CORRESPONDING SLAM SOURCE 
C CODE ON FILE 20.
C 
C SUBROUTINES CALLED:
C BLDOUT--BUILD THE SLAM OUTPUT
C CHKBLK--VALIDATE BALK ROUTES
C CHKUSD--ENSURE ALL NODES ARE USED
C RALK--PROCESS BALK CARDS
C MESS--PROCESS MESS CARDS
C NODE--PROCESS NODE CARDS
C PROC--PROCESS PROC CARDS
C PATH--PROCESS PATH CARDS
C TERM--PROCESS TERM CARDS
C XMIT--PROCESS XMIT CARDS
C READ--READ INPUT FILE
C 
C FUNCTIONS CALLED:
C NUMNOD--RETURNS THE NODE'S NUMBER
C NUMTRM--RETURNS THE TERM'S NUMBER
C 
C INPUT ARGUMENTS:
C NONE
C 
C OUTPUT ARGUMENTS:
C NONE
C 
C INPUTS VIA COMMON:
C IFATAL
C NXTNOD
C NXTTRM
C NXTMSG
C PARM1
C PARM2
C PARM3
C PARM4
C PARM5
C CRDTYP
C LABEL
C LABEL2
C DSTNAM
C 
C OUTPUTS VIA COMMON:
C IFATAL
C NXTNOD
C NXTTRM
C NXTMSG
C MAXNOD
C MAXTRM
C MAXMSG
C PTHMAT
C NODNAM
C TRMNAM
C DSTNAM
C 
C WORKING VARIABLES:
C I--DO LOOP COUNTER
PARAMETERS FOR DIMENSION SIZES:
NONOD--MAXIMUM NUMBER OF NODES
NOTRM--MAXIMUM NUMBER OF TERMINATE NODES
NOMSG--MAXIMUM NUMBER OF MESSAGES

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
& BALKS*4, DSTNAM*5

DESCRIPTION OF COMMON BLOCK /COM1/:
**COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM, & MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, Nomsg), & CREATE(4, Nomsg), LASNOD(Nomsg), NODSTR(Nomsg), & PARM1, PARM2, PARM3, PARM4, PARM5, & DSTPRC(Nomsg, NONOD, 4), DSTXMT(Nomsg, NONOD, 4)

DESCRIPTION OF COMMON BLOCK /COM1/:

C NODNAM(I)--NAME OF NODE I
C DIST(I)--NAME OF THE MESSAGE DISTRIBUTION FOR MESSAGE I
C TRMNAM(I)--NAME OF TERMINATE NODE I
C CRDTYP--CARD TYPE READ FROM INPUT DECK
C LABEL--1ST CHARACTER DATA FIELD READ FROM INPUT DECK
C LABEL2--2ND CHARACTER DATA FIELD READ FROM INPUT DECK
C BALKS(I)--
  IF BLANK, NODE I DOES NOT BLOCK OR BALK
  IF 'BLOK', NODE I IS BLOCKED
  OTHERWISE, NODE NAME THAT NODE I BALKS TO
C DSTNAM--LIST OF VALID DISTRIBUTION NAMES

**COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP, & LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

C PREPARE INPUT FILE
REWIND 10

C INITIAL VALUES
IFATAL = 0
NXTNOD = 0
NXTTRM = 0
NXTMSG = 0
MAXNOD = NONOD
MAXTRM = NOTRM
MAXMSG = NOMSG

C VALID DISTRIBUTION NAMES
DSTNAM(1) = 'EXPOH'
DSTNAM(2) = 'NPSSN'
DSTNAM(3) = 'BETA'
DSTNAM(4) = 'GAMA'
DSTNAM(5) = 'RLOGH'
DSTNAM(6) = 'RNORM'
DSTNAM(7) = 'UNFRM'
DSTNAM(8) = 'TRIAG'
DSTNAM(9) = 'CONST'
DO 1 1 = 1, MAXMSG
DO 1 J = 1, MAXNOD
C ZERO OUT PTIMAT (I.E., ESTABLISH NO PATHS EXIST FOR ANY MESSAGE).
1 PTIMAT(J, I) = 0.0
DO 2, I = 1, MAXTRM
C INITIALIZE ALL TERMINATE NODES TO BLANK (UNUSED).
2 TRMNAM(I) = ' '
DO 3, I = 1, MAXNOD

18
C INITIALIZE ALL NODES TO BLANK (UNUSED).
3 NODNAM(I) = ' ' 
PRINT *, ' ' 
PRINT *, ' ' 
PRINT *, ' SLAM PRE-PROCESSOR' 
PRINT *, ' ' 
PRINT *, ' SUMMARY OF PROGRAM INPUTS:'
C
C ------- PROCESS NODE CARDS ------- 
C
PRINT *, ' ' 
20 CALL READ
C NODE CARDS MUST BE FIRST.
IF (CRDTYP .EQ. 'NODE') THEN
C A NODE CARD WAS FOUND--PROCESS IT.
10 CALL NODE (LABEL, PARM2, PARM3)
CALL READ
C ANOTHER NODE CARD?
IF (CRDTYP .EQ. 'NODE') THEN
GO TO 10
ENDIF
ELSE
C SOMETHING OTHER THAN A NODE CARD WAS FIRST.
PRINT *, 'ERROR--NODE CARDS MUST BE FIRST.' 
IFATAL = IFATAL + 1
GO TO 20
C
C ------- PROCESS TERM CARDS ------- 
C
ENDIF
CONTINUE
C FOLLOWING NODE CARDS MUST BE TERM CARDS.
IF (CRDTYP .EQ. 'TERM') THEN
C A TERM CARD WAS FOUND--PROCESS IT.
40 CALL TERM(LABEL)
CALL READ
C ANOTHER TERM CARD?
IF (CRDTYP .EQ. 'TERM') THEN
GO TO 40
ENDIF
ELSE
C SOMETHING OTHER THAN A TERM CARD WAS FOUND.
PRINT *, 'ERROR--TERM CARDS MUST FOLLOW NODE CARDS.' 
IFATAL = IFATAL + 1
CALL READ
GO TO 30
ENDIF
C
C ------- PROCESS MESSAGES ------- 
C
C
C EACH MESSAGE CONSISTS OF A MESS CARD, FOLLOWED BY A PROC (OPTIONAL),
C AN XMIT (OPTIONAL), AND A PATH CARD. CONTINUE PROCESSING MESSAGES
C UNTIL A NON-MESS, PROC, XMIT OR PATH CARD IS FOUND.
C
C FOLLOWING TERM CARDS MUST BE A MESS CARD.
50  IF (CRDTYP .NE. 'MESS') THEN
C SOMETHING OTHER THAN A MESS CARD WAS FOUND.
   PRINT *, 'ERROR--MESS CARD MUST FOLLOW TERM CARDS.'
   IFATAL = IFATAL + 1
   CALL READ
   GO TO 50
ELSE
A MESS CARD WAS FOUND--PROCESS IT.
60  CALL MESS(LABEL, PARM1, PARM2, PARM3, PARM4, PARM5, LABEL2)
   CALL READ
C FOLLOWING A MESS CARD CAN BE A PATH CARD, A PROC
C CARD, OR AN XMIT CARD.
C  IF (CRDTYP .EQ. 'PROC') THEN
C   A PROC CARD WAS FOUND--PROCESS IT.
C     CALL PROC (LABEL, PARM3, PARM4, PARM1)
C     CALL READ
ELSE
NO PROC CARD WAS FOUND--ESTABLISH DEFAULT PROCESS TIME
C     CALL PROC('CONST', 0.0, 0.0, 0.0)
ENDIF
C  IF (CRDTYP .EQ. 'XMIT') THEN
C AN XMIT CARD WAS FOUND--PROCESS IT.
C   CALL XMIT(LABEL, PARM3, PARM4, PARM1)
C   CALL READ
ELSE
NO XMIT CARD WAS FOUND--ESTABLISH DEFAULT TRANSMIT TIME
C   CALL XMIT('CONST', 0.0, 0.0, 0.0)
ENDIF
C A PATH CARD MUST BE NEXT.
C  IF (CRDTYP .NE. 'PATH') THEN
C NO PATH CARD FOUND.
C     NXTMSG = NXTMSG - 1
C     PRINT *, 'ERROR--MISSING PATH CARD. SKIPPING TO NEXT MESS.'
    CALL READ
C GO TO BEGINNING OF NEXT MESSAGE.
C  IF (CRDTYP .EQ. 'MESS') THEN
C    GO TO 60
ELSE
C    GO TO 70
ENDIF
ELSE
A PATH CARD WAS FOUND--PROCESS IT.
80  CALL PATH
C  IF NUMTRM(LABEL) = 0, THEN THE LAST NODE SPECIFIED
C WAS NOT A TERMINATE NODE. THEREFORE, THE NEXT CARD
C MUST BE EITHER PROC, XMIT OR PATH TO CONTINUE
C DESCRIBING THIS MESSAGE PATH.
C  IF (NUMTRM(LABEL) .EQ. 0) THEN
C   LAST NODE WAS NOT A TERMINATE NODE.
20
CALL READ
IF (CRDTYP .EQ. 'PROC') THEN
  A PROC CARD WAS FOUND--PROCESS IT.
  CALL PROC(LABEL, PARM3, PARM4, PARM1)
  CALL READ
ELSE
  NO PROC CARD WAS FOUND--ESTABLISH DEFAULT PROCESS TIME
  CALL PROC('CONST', 0.0, 0.0, 0.0)
ENDIF
IF (CRDTYP .EQ. 'XMIT') THEN
  AN XMIT CARD WAS FOUND--PROCESS IT.
  CALL XMIT(LABEL, PARM3, PARM4, PARM1)
  CALL READ
ELSE
  NO XMIT CARD WAS FOUND--ESTABLISH DEFAULT TRANSMIT TIME.
  CALL XMIT('CONST', 0.0, 0.0, 0.0)
ENDIF
IF (CRDTYP .EQ. 'PATH') THEN
  A PATH CARD WAS FOUND--PROCESS IT.
  GO TO 80
ENDIF
C NO PATH CARD WAS FOUND.
PRINT *, 'ERROR--MESSAGE MUST END ON A TERMINATE NODE.'
IFATAL = IFATAL + 1
ELSE
  THE LAST NODE WAS A TERMINATE NODE.
  CALL READ
ENDIF
ENDIF
ENDIF
C FOLLOWING THE LAST PATH CARD OF A MESSAGE MAY BE ANOTHER
C MESS CARD, A BALK CARD, OR AN END CARD.
IF (CRDTYP .EQ. 'MESS') THEN
  ANOTHER MESSAGE IS BEING DEFINED.
  GO TO 50
ENDIF
C SAVE NXTNOD AND NXTMSG FOR LATER USE.
ISAVND = NXTNOD
ISAVMS = NXTMSG
IF (CRDTYP .NE. 'BALK') THEN
  A BALK ROUTE IS NOT BEING DEFINED.
  GO TO 500
ENDIF
C -- PROCESS BALK ROUTES --
C EACH BALK ROUTE CONSISTS OF A BALK CARD, FOLLOWED BY A PROC(OPTIONAL
C AN XMIT (OPTIONAL), AND A PATH CARD. CONTINUE PROCESSING BALK ROUTE
C UNTIL A NON-BALK, PROC, XMIT OR PATH CARD IS FOUND

21
C ESTABLISH THE NXTNOD NUMBER FOR THIS BALK ROUTE.
400 NXTNOD = NUMNOD(LABEL)
   IF (NXTNOD .EQ. 0) THEN
      PRINT *, 'ERROR--INVALID BALK NODE (SKIPPING TO NEXT BALK.)'
      PRINT *, 'CORRECT NODE NAME OR INSERT CORRESPONDING ',
      'TERM CARD.'
      IFATAL = IFATAL + 1
   ENDIF
   CALL READ
   IF (CRDTYPE .EQ. 'BALK') THEN
      GO TO 400
   ELSE
      GO TO 402
   ENDIF
C ESTABLISH THE NXTMSG NUMBER FOR THIS BALK ROUTE.
C NXTMSG IS THE MESSAGE NUMBER OF THE MESSAGE FOR WHOM A BALK
C ROUTE IS BEING DEFINED.
C NXTMSG = PARM1
   IF (NXTMSG .GT. ISAVMS .OR. NXTMSG .LE. 0) THEN
      PRINT *, 'ERROR--INVALID MESSAGE NUMBER (SKIPPING TO NEXT ',
      'BALK).'
      IFATAL = IFATAL + 1
   ENDIF
   CALL READ
   IF (CRDTYPE .EQ. 'BALK') THEN
      GO TO 400
   ELSE
      GO TO 401
   ENDIF
C A VALID BALK CARD WAS FOUND--PROCESS IT.
CALL BALK
CALL READ
C FOLLOWING A BALK CARD MAY BE EITHER A PROC CARD, XMIT
C CARD, OR PATH CARD.
410 IF (CRDTYPE .EQ. 'PROC') THEN
C A PROC CARD WAS FOUND--PROCESS IT.
   CALL PROC(LABEL, PARM3, PARM4, PARM1)
   CALL READ
ELSE
C NO PROC CARD WAS FOUND--ESTABLISH DEFAULT PROCESS TIME.
   CALL PROC ('CONST', 0.0, 0.0, 0.0)
ENDIF
IF (CRDTYPE .EQ. 'XMIT') THEN
C AN XMIT CARD WAS FOUND--PROCESS IT.
   CALL XMIT(LABEL, PARM3, PARM4, PARM1)
   CALL READ
ELSE
C NO XMIT CARD WAS FOUND--ESTABLISH DEFAULT TRANSMIT TIME.
   CALL XMIT('CONST', 0.0, 0.0, 0.0)
ENDIF
IF (CRDTYPE .NE. 'PATH') THEN
C THIS BALK ROUTE CONTAINED NO PATH.
   PRINT *, 'ERROR--MISSING BALK PATH CARD. SKIPPING TO NEXT ',
   'BALK.'
ENDIF
IFATAL = IFATAL + 1

600 CALL READ
C ANOTHER BALK?
   IF (CRDTYP .EQ. 'BALK') THEN
     GO TO 400
   ELSE
     GO TO 600
   ENDIF
ELSE
   A PATH CARD WAS FOUND--PROCESS IT.
   CALL PATH
ENDIF
C CALL READ
C IS THE NEXT CARD A PROC OR PATH OR XMIT (I.E., A CONTINUATION
C OF THIS BALK PATH)?
   IF (CRDTYP .EQ. 'PROC' .OR. CRDTYP .EQ. 'XMIT' .OR.
6 CRDTYP .EQ. 'PATH') THEN
     GO TO 410
   ENDIF
C FOLLOWING THE LAST PATH CARD OF A BALK ROUTE MAY BE
C ANOTHER BALK CARD OR AN END CARD.
   IF (CRDTYP .EQ. 'BALK') THEN
     C ANOTHER BALK CARD WAS FOUND.
     GO TO 400
   END
C
C - - - END OF DATA--VALIDATE AND PRODUCE SLAM OUTPUT - - - - - - -
C ENDIF
500 CONTINUE
C THE NEXT CARD MUST BE AN END.
   IF (CRDTYP .NE. 'END') THEN
     PRINT *, 'ERROR--END CARD EXPECTED.'
     IFATAL = IFATAL + 1
   ENDIF
C RESTORE NXTMSG AND NXTNOD TO THEIR PREVIOUS VALUES.
   NXTMSG = ISAVMS
   NXTNOD = ISAVND
C VERIFY BALK ROUTES.
   IF (IFATAL .EQ. 0) THEN
     CALL CHKBLK
   ELSE
     PRINT *, 'WARNING--BALK ROUTE VERIFICATION NOT ATTEMPTED DUE', &
8 ' TO PREVIOUS ERRORS.'
   ENDIF
C CALL CHKUSD
   IF (IFATAL .EQ. 0) THEN
     IF NO ERRORS DETECTED, PRODUCE SLAM OUTPUT.
     PRINT *, 'NO ERRORS DETECTED. SLAM SOURCE WILL BE CONSTRUCTED.'
     CALL BLDOUT
   ELSE
     PRINT *, IFATAL, ' ERRORS DETECTED. SLAM SOURCE WILL NOT BE '.
SUBROUTINE BALK

THIS SUBROUTINE HANDLES BALK CARDS. WHEN A BALK IS ENCOUNTERED, LASNOD (THE PREVIOUS NODE NUMBER) IS INITIALIZED TO THE FIRST NODE IN THE BALK PATH.

CALLED BY:
MAIN

SUBROUTINES CALLED:
NONE

FUNCTIONS CALLED:
NUMNOD

INPUT ARGUMENTS:
NONE

OUTPUT ARGUMENTS:
NONE

INPUTS VIA COMMON:
IFATAL
NXTMSC
LABEL

OUTPUT VIA COMMON:
IFATAL
LASNOD

WORKING VARIABLES:
NUMN--NODE NUMBER

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAME*4
6 BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSC, MAXNOD, MAXTRM,
6 MAXMSC, NODCAP(2, NONOD), PTHMAT(NONOD, NOMSG),
6 CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
6 PARM1, PARM2, PARM3, PARM4, PARM5,
6 DSTPRC(NOMSG, NONOD, 4), DSTMXT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAME(NOTRM), CRDTYP,
6 LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

FIND THE NODE NUMBER OF THE BALK NODE (NUMN MUST BE > 0 TO BE VALID).
NUMN = NUMNOD(LABEL)
IF (NUMN .GT. 0) THEN

INITIALIZE LASNOD TO THE 1ST NODE IN THIS BALK PATH.
LASNOD(NXTMSC) = NUMN
ELSE
PRINT *, 'ERROR--ILLEGAL BALK NODE NAME.'
IFATAL = IFATAL + 1
ENDIF
RETURN
END
SUBROUTINE BLDOUT

C
C THIS SUBROUTINE PRODUCES THE SLAM SOURCE CODE, WHICH IS
C WRITTEN TO FILE 20.
C
C CALLED BY:
C MAIN
C SUBROUTINES CALLED:
C NONE
C FUNCTIONS CALLED:
C NUMDST
C INPUT ARGUMENTS:
C NONE
C OUTPUT ARGUMENTS:
C NONE
C INPUTS VIA COMMON:
C NXTNOD
C NXTTRM
C NXTMSG
C NODCAP
C PTIMAT
C CREATE
C NODSTR
C DSTPRC
C DSTXMT
C NODNAM
C DIST
C TRNMAM
C BALKS
C DSTNAM
C OUTPUTS VIA COMMON:
C NONE
C WORKING VARIABLES:
C BRNAME--BRANCH NODE NAME
C I--DO LOOP COUNTER
C ICNT--GENERATED LINE LABEL NUMBER
C IMAX--INTEGER REPRESENTATION OF STARTING TIME OF MESSAGE
C IMIN--INTEGER REPRESENTATION OF NUMBER OF MESSAGE RELEASES
C ISTRT--LINE LABEL NUMBER
C J--DO LOOP COUNTER
C NUM--THE NEXT NODE NUMBER IN THE PATH
C NUMD--DISTRIBUTION NUMBER

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER BRNAME*4
CHARACTER CRDIYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDIYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)
REWIND 20

C WRITE SLAM GEN CARD.
WRITE (20, 10)
10 FORMAT ('GEN, SLAMPP, SLAM, 09/19/1982, 1, ..., 72;')
WRITE (20, 15) NXTNOD
C WRITE LIM CARD.
15 FORMAT ('LIM, ', I4, ',', 2, 500;')
WRITE (20, 20)
20 FORMAT ('NETWORK;')
C WRITE RESOURCE CARD FOR EACH NODE.
DO 300, 1 - 1, NXTNOD
300 WRITE (20, 301) NODNAM(I), NODCAP(2, I), I
301 FORMAT ('RESOURCE/', A4, '(', F10.5, ',', 15, ');')
C WRITE MESSAGE CREATION CARDS. EACH MESSAGE CONSISTS OF A
C CREATE CARD, ASSIGN CARD, AND AN ACT CARD.
DO 900 1 - 1, NXTMSC
C STARTING TIME FOR THIS MESSAGE.
IMIN = CREATE (4, I)
C NUMBER OF MESSAGE RELEASES.
IMAX = CREATE (5, I)
C NUMBER OF THE CREATION DISTRIBUTION RATE.
NUMD = NUMDIST(DIST(I))
C EACH DISTRIBUTION RATE REQUIRES A CERTAIN NUMBER OF INPUT
C PARAMETERS. THE FOLLOWING SELECTS THE APPROPRIATE WRITE
C STATEMENT FOR THE DISTRIBUTION.
IF (NUMD .EQ. 9) THEN
C CONSTANT CREATION RATE.
WRITE (20, 800) CREATE(I, I), IMIN, IMAX
800 FORMAT ('CREATE', F10.5, ',', 15, ');')
ELSE IF (NUMD .LE. 2) THEN
C FOR EXPON AND NPSN (REQUIRING ONE PARAMETER).
WRITE (20, 805) DIST(I), CREATE(I, I), IMIN, IMAX
805 FORMAT ('CREATE', A5, '(', F10.5, '),', 15, ');')
ELSE IF (NUMD .LE. 7) THEN
C FOR BETA, GAMA, RLOGN, RNORM AND UNFRM (REQUIRING TWO
C PARAMETERS).
WRITE (20, 810) DIST(I), CREATE(I, I), CREATE(2, I), IMIN, IMAX
810 FORMAT ('CREATE', A5, '(', F10.5, ',', F10.5, '),', 15, ',', 15, ');')
ELSE
C FOR TRIAC (REQUIRING THREE PARAMETERS).
WRITE (20, 820) DIST(I), CREATE(I, I), CREATE(2, I), CREATE(3, I), IMIN, IMAX
820 FORMAT ('CREATE', A5, '(', 2(F10.5, '),', 15, ',', 15, ');')
ENDIF
C EACH MESSAGE IS ASSIGNED A SEQUENTIAL MESSAGE NUMBER SO
C THAT DIFFERENT MESSAGES CAN BE DISTINGUISHED FROM EACH
C OTHER THROUGHOUT THE NETWORK.
WRITE (20, 30) I
30 FORMAT ('ASSIGN, ATRIB(2) = ',', I4, ';')
ACT TO BRANCH TO 1ST NODE IN THE PATH.
WRITE (20, 35) NODNAM(NODSTR(I))

35 FORMAT (' ACT,,','A4',';')

CONTINUE

C ESTABLISH BEGINNING LABEL NUMBER FOR PROGRAM GENERATED LABELS.
ICNT = 101

C GENERATE THE SLAM OUTPUT NECESSARY TO DESCRIBE EACH NODE.
C THIS WILL CONSIST OF AN AWAIT CARD, BRANCH CARDS (ACT) FOR
C EACH MESSAGE, PLUS A SET OF PROCESS AND TRANSMIT ACTIVITIES
C FOR EACH MESSAGE USING THIS NODE.
DO 950 I = 1, NXTNOD

C THREE TYPES OF AWAIT STATEMENTS ARE POSSIBLE: AN AWAIT WITHOUT
C ANY BLOCKING OR BALKING, AN AWAIT WITH BLOCKING, OR AN AWAIT
C WITH BALKING.
IF (BALKS(I) .EQ. '') THEN
  NO BLOCKING OR BALKING.
  WRITE (20, 40) NODNAM(I), I, NODCAP(I, I), NODNAM(I)
ELSE IF (BALKS(I) .EQ. 'BLOK') THEN
  BLOCKING.
  WRITE (20, 41) NODNAM(I), I, NODCAP(I, I), NODNAM(I)
ELSE
  BALKING.
  WRITE (20, 42) NODNAM(I), I, NODCAP(I, I), NODNAM(I),
& BALKS(I)
ENDIF

C PLACE A DUMMY ACTIVITY FOLLOWING THE AWAIT.
38 FORMAT (' ACT;')
WRITE (20,29)

C GOON TO SPECIFY ONLY ONE BRANCH TO BE TAKEN.
29 FORMAT (' GOON,1;')

C SAVE THE NUMBER OF THE FIRST PROGRAM GENERATED LABEL.
ISTRT = ICNT

DO 925 J = 1, NXTMSG

C GET THE NODE NUMBER OF THE I-TH NODE IN PATH J.
NUM = PTRMAT (I, J)
C IF 0, MESSAGE I DOES NOT USE PATH J.
IF (NUM .NE. 0) THEN
  CONSTRUCT SLAM BRANCH TO THAT PORTION OF THE SLAM CODE
  WHICH WILL HANDLE THE PROCESS AND TRANSMIT TIMES FOR THIS
  MESSAGE.
  WRITE (20, 500) J, ICNT
500 FORMAT (' ACT,,ATRIB(2).EQ.,I4,','L',I3,';')
C INCREMENT PROGRAM GENERATED LABEL NUMBER.
ICNT = ICNT + 1
ENDIF
CONTINUE

925 CONTINUE

C GENERATE A PROCESS AND TRANSMIT ACTIVITY PAIR FOR EACH MESSAGE
C THAT USES NODE J.
DO 926 J = 1, NXTMSG

C GET THE NODE NUMBER OF THE I-TH NODE IN PATH J.
NUM = PNUMAT(I, J)
C IF 0, MESSAGE 1 DOES NOT USE PATH J.
C IF (NUM .NE. 0) THEN
  WRITE (20, 501) ISTRT
C CONSTRUCT DUMMY COON WITH PROGRAM GENERATED LABEL.
  501 FORMAT ('L', 13, ' COON;')
C INCREMENT PROGRAM GENERATED LABEL NUMBER.
  ISTRT = ISTRT + 1
C EACH DISTRIBUTION RATE Requires A CERTAIN NUMBER OF INPUT
C PARAMETERS. THE FOLLOWING SELECTS THE APPROPRIATE WRITE
C STATEMENT FOR THE DISTRIBUTION.
C
C IF THE DISTRIBUTION IS A CONSTANT ZERO, DO NOT WRITE ANYTHING.
  IF (DSTPRC(J, I, 1) .EQ. 9.0 .AND.
    DSTPRC(J, I, 2) .EQ. 0.0) THEN
    GO TO 600
  ENDIF
C CONSTANT PROCESS TIME.
  IF (DSTPRC(J, I, 1) .EQ. 9.0) THEN
    WRITE (20, 502) DSTPRC(J, I, 2)
    502 FORMAT (6X, 'ACT,', FIO.5, ';')
  ELSE IF (DSTPRC(J, I, 1) .GE. 2.0) THEN
C FOR EXPON AND NPSN (REQUIRING ONE PARAMETER).
    WRITE (20, 503) DSTNAM(INT(DSTPRC(J, I, 1))),
      DSTPRC(J, I, 2)
    503 FORMAT (6X, 'ACT,', AS, '('
      F10.5, ',';')
  ELSE IF (DSTPRC(J, I, 1) .LE. 7.0) THEN
C FOR BETA, GAMMA, LOGN, RNSN AND UNFRN (REQUIRING TWO
C PARAMETERS).
    WRITE (20, 504) DSTNAM(INT(DSTPRC(J, I, 1))),
      DSTPRC(J, I, 2), DSTPRC(J, I, 3)
    504 FORMAT (6X, 'ACT,', AS, '('
      F10.5, ',', F10.5, ');')
  ELSE
C FOR TRIAC (REQUIRING THREE PARAMETERS).
    WRITE (20, 505) DSTNAM(INT(DSTPRC(J, I, 1))),
      DSTPRC(J, I, 2), DSTPRC(J, I, 3), DSTPRC(J, I, 4)
    505 FORMAT (6X, 'ACT,', AS, '('
      2(FIO.5, ','), F10.5, ');')
  ENDIF
C PLACE A DUMMY COON BETWEEN PROCESS AND TRANSMIT ACTIVITIES.
  WRITE (20, 506)
  506 FORMAT (' COON;')
C FInd THE NODE NAME OF THE NEXT NODE IN THE PATH. IF
C NUM > 0, THEN THE NEXT NODE IS A NODE. IF < 0, THEN THE
C NEXT NODE IS A TERMINATE NODE.
  IF (NUM .GT. 0) THEN
    BRNAME = NODNAM(NUM)
  ELSE
    BRNAME = TRMNAM(-NUM)
  ENDIF
C OBTAIN THE NUMBER OF SERVERS FOR THIS TRANSMIT ACTIVITY.

EACH DISTRIBUTION RATE REQUIRES A CERTAIN NUMBER OF INPUT PARAMETERS. THE FOLLOWING SELECTS THE APPROPRIATE WRITE STATEMENT FOR THE DISTRIBUTION.

IF THE DISTRIBUTION IS A CONSTANT ZERO, DO NOT WRITE ANYTHING.

IF (DSTXMT(J, I, 1) .EQ. 9.0) .AND. DSTXMT(J, I, 2) .EQ. 0.0) THEN
   GO TO 601
ENDIF

IF (DSTXMT(J, I, 1) .EQ. 9.0) THEN
   CONSTANT PROCESS TIME.
   WRITE (20, 502) DSTXMT(J, I, 2)
ENDIF

ELSE IF (DSTXMT(J, I, 1) .LE. 2.0) THEN
   FOR EXPON AND NPSSN (REQUIRING ONE PARAMETER).
   WRITE (20, 503) DSTNAM(INT(DSTXMT(J, I, 1))), DSTXMT(J, I, 2)
   ELSE IF (DSTXMT(J, I, 1) .LE. 7.0) THEN
      FOR BETA, GAMMA, RLOGN, RNORM AND UNFRM (REQUIRING TWO PARAMETERS).
      WRITE (20, 504) DSTNAM(INT(DSTXMT(J, I, 1))), DSTXMT(J, I, 2), DSTXMT(J, I, 3)
      ELSE
         FOR TRIAC (REQUIRING THREE PARAMETERS).
         WRITE (20, 505) DSTNAM(INT(DSTXMT(J, I, 1))), DSTXMT(J, I, 2), DSTXMT(J, I, 3), DSTXMT(J, I, 4)
      ENDIF
601 CONTINUE

WRITE (20, 303) NODNAM(I)
303 FORMAT ('FREE,' , A4, ';')
WRITE (20, 35) BRNAME

926 CONTINUE
950 CONTINUE
DO 960 I = 1, NXTTRM
   PRINT EACH OF THE TERMINATE NODE CARDS.
   WRITE (20, 50) TRMNAM(I), TRMNAM(I)
50 FORMAT (A4, 2X, 'COLCT,INT(1),EXIT INTRVL ', A4, ';')
   WRITE (20, 55)
55 FORMAT ('TERM;')
960 CONTINUE
WRITE (20, 59)
59 FORMAT ('END;')
WRITE (20, 60)
60 FORMAT ('INIT,0;')
WRITE (20, 65)
65 FORMAT ('FIN;')
RETURN
END
SUBROUTINE CHKRLK

THIS SUBROUTINE VERIFIES THAT ALL BALK PATHS ARE VALID, I.E., ALL PATHS END AT A TERMINATE NODE AND NO PATH CONTAINS A CYCLE.

CALLED BY:
MAIN
SUBROUTINES CALLED:
NONE
FUNCTIONS CALLED:
NUMNOD
INPUT ARGUMENTS:
NONE
OUTPUT ARGUMENTS:
NONE
INPUTS VIA COMMON:
IFATAL
NXTNOD
NXTMSG
PTIMAT
NODNAM
BALKS
OUTPUTS VIA COMMON:
IFATAL
WORKING VARIABLES:
I--DO LOOP COUNTER
J--DO LOOP COUNTER
NEXT--NEXT NODE NUMBER ON THE PATH
NOSTEP--COUNT OF THE NUMBER OF NODES IN A PATH
NUM--BALK NODE NUMBER

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4,
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)
DO 10 I = 1, NXTNOD
IF BALK(I) IS BLANK OR 'BLOK', THEN A BALK ROUTE WAS NOT SPECIFIED FOR NODE I AND, THEREFORE, NEED NOT BE CHECKED.
IF (BALKS(I) .NE. ' ' .AND. BALKS(I) .NE. 'BLOK') THEN
FIND THE NODE NUMBER THAT NODE I WILL BALK TO.
NUM = NUMNOD(BALKS(I))
DO 20 J = 1, NXTMSG
IF PTIMAT(I, J) = 0, THEN MESSAGE J DOES NOT USE NODE I AND CAN BE IGNORED. OTHERWISE, FOR EACH MESSAGE J IN NODE I INSURE THAT BALK NODE NUM HAS A PATH FOR MESSAGE J (I.E., IF MESSAGE J APPEARS IN NODE I, MESSAGE J MUST ALSO APPEAR IN NODE NUM).
IF (PTIMAT(I, J) .NE. 0) THEN
C NOSTEP WILL COUNT HOW MANY NODES ARE IN THE MESSAGE'S PATH.  
NOSTEP = 0  
C FIND THE NEXT NODE NUMBER IN THE BALK PATH FOR MESSAGE J.  
NEXT = PTHMAT(NUM, J)  
30 IF (NEXT .EQ. 0) THEN  
C BALK PATH FOR MESSAGE J WAS NOT SPECIFIED OR WAS  
C INCOMPLETE (DID NOT END ON A TERMINATE NODE).  
PRINT 50, J, NODNAM(I)  
PRINT */,' SUPPLY BALK PATH, OR MAKE SURE PATH ',  
& 'ENDS AT A TERMINATE NODE.'  
IFATAL = IFATAL + 1  
ELSE  
C IF > 0, NEXT NODE IS A REGULAR NODE. IF < 0, NEXT  
C NODE IS A TERMINATE NODE (MEANING THIS PATH IS  
C SUCCESSFULLY COMPLETED).  
IF (NEXT .GT. 0) THEN  
C FIND THE NEXT NODE NUMBER IN THE BALK PATH FOR MESSAGE J.  
NEXT = PTHMAT(NEXT, J)  
C INCREMENT THE NUMBER OF NODES IN THIS BALK PATH.  
NOSTEP = NOSTEP + 1  
C IF MORE STEPS IN THE PATH WAS USED THAN NODES WERE  
C DEFINED, THEN A CYCLE MUST EXIST FOR THIS PATH  
C (MESSAGE J LOOPS BACK UPON ITSELF).  
IF (NOSTEP .GT. NXTNOD) THEN  
PRINT 55, J, NODNAM(I)  
IFATAL = IFATAL + 1  
GO TO 20  
ENDIF  
ENDIF  
GO TO 30  
ENDIF  
ENDIF  
ENDIF  
ENDIF  
CONTINUE  
ENDIF  
10 CONTINUE  
50 FORMAT (' ERROR--INCOMPLETE BALK PATH FOR MESSAGE ', I4,  
& ' FROM NODE ', A4, '.')  
55 FORMAT (' ERROR--THE BALK PATH FOR MESSAGE ', I4,  
& ' CONTAINS A CYCLE FROM NODE ', A4, '.')  
RETURN  
END
SUBROUTINE CIIKUSD

THIS SUBROUTINE VERIFIES THAT EACH NODE SPECIFIED
BY A NODE CARD IS USED IN AT LEAST ONE PATH.

CALLED BY:
MAIN

SUBROUTINES CALLED:
NONE

FUNCTIONS CALLED:
NONE

INPUT ARGUMENTS:
NONE

OUTPUT ARGUMENTS:
NONE

INPUTS VIA COMMON:
IFATAL
NXTNOD
NXTMSG
PTHMAT
NODNAM

OUTPUTS VIA COMMON:
IFATAL

WORKING VARIABLES:
I--DO LOOP COUNTER
J--DO LOOP COUNTER

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4.
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTHMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

SEARCH EACH NODE I LOOKING FOR A MESSAGE J THAT USES NODE I
(I.E., PTHMAT(I, J) .NE. 0).
DO 20, I = 1, NXTNOD
DO 10, J = 1, NXTMSG
IF (PTHMAT(I, J) .NE. 0.0) THEN
  THE NODE WAS USED.
  GO TO 20
ENDIF
CONTINUE
10 PRINT *, 'ERROR--NODE ', NODNAM(I), ' IS NOT USED IN A PATH.'
PRINT *, ' REMOVE CORRESPONDING NODE CARD.'
IFATAL = IFATAL + 1
20 CONTINUE
RETURN
END
SUBROUTINE MESS (LABL, PRM1, PRM2, PRM3, PRM4, & PRM5, LABL2)

C THIS SUBROUTINE HANDLES MESS CARDS BY SAVING THE NECESSARY
C MESSAGE INFORMATION.
C
C CALLED BY:
C    MAIN
C SUBROUTINES CALLED:
C    NONE
C FUNCTIONS CALLED:
C    NUMDST
C    NUMNOD
C
INPUT ARGUMENTS:
C    LABL--MESSAGE DISTRIBUTION NAME
C    PRM1--DISTRIBUTION PARAMETER 1
C    PRM2--DISTRIBUTION PARAMETER 2
C    PRM3--DISTRIBUTION PARAMETER 3
C    PRM4--MESSAGE START TIME
C    PRM5--NUMBER OF MESSAGE RELEASES
C    LABL2--NAME OF FIRST NODE IN PATH
C
OUTPUT ARGUMENTS:
C    NONE
C
INPUTS VIA COMMON:
C    IFATAL
C    NXTMSG
C    MAXNOD
C
OUTPUTS VIA COMMON:
C    IFATAL
C    NXTMSG
C    CREATE
C    LASNOD
C    NODSTR
C    DIST
C
WORKING VARIABLES:
C    NUMN--NODE NUMBER
C
PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER LABL*5, LABL2*4
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4.
& BALKS*4, DSTMN*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTMN(9)
C INCREMENT MESSAGE COUNT.
C    NXTMSG = NXTMSG + 1
C INSURE NXTMSG STILL IN RANGE.
C IF (NXTMSG .GT. MAXMSG) THEN
C    PRINT *, 'ERROR--TOO MANY MESSAGES DEFINED.'
C    IFATAL = IFATAL + 1

34
NXTMSC = MAXMSG
LASNOD(NXTMSG) = 1
GO TO 999
ENDIF
C
VERIFY LABL IS A VALID DISTRIBUTION NAME.
! IF (NUMDST(LABL) .GT. 0) THEN
C
VALID NAME, SAVE IT.
DIST(NXTMSG) = LABL
ELSE
PRINT *, 'ERROR--ILLEGAL DISTRIBUTION SPECIFIED.'
IFATAL = IFATAL + 1
ENDIF
C
SAVE 1ST PARAMETER OF DISTRIBUTION
CREATE(1, NXTMSG) = PRM1
C
SAVE 2ND PARAMETER OF DISTRIBUTION.
CREATE(2, NXTMSG) = PRM2
C
SAVE 3RD PARAMETER OF DISTRIBUTION.
CREATE(3, NXTMSG) = PRM3
C
SAVE STARTING TIME OF THIS MESSAGE.
CREATE(4, NXTMSG) = PRM4
C
SAVE THE NUMBER OF MESSAGE RELEASES.
CREATE(5, NXTMSG) = PRM5
C
FIND THE NODE NUMBER OF THE FIRST NODE IN THIS MESSAGE.
NUMN = NUMNOD(LABL)2
IF (NUMN .GE. 0) THEN
C
VALID NODE NUMBER--INITIALIZE THE LAST NODE IN THE PATH
C (LASNOD) AND THE FIRST NODE IN THE PATH (NODSTR) TO NUMN.
LASNOD(NXTMSG) = NUMN
NODSTR(NXTMSG) = NUMN
ELSE
PRINT *, 'ERROR--UNDEFINED NODE.'
PRINT *, 'CORRECT NODE NAME OR INSERT CORRESPONDING ',
& 'NODE CARD.'
LASNOD(NXTMSG) = 1
IFATAL = IFATAL + 1
ENDIF
999 RETURN END
SUBROUTINE NODE (LABI, PRM1, PRM2)

THIS SUBROUTINE HANDLES NODE CARDS BY SAVING THE NODE INFORMATION

CALLED BY:
MAIN

SUBROUTINES CALLED:
NONE

FUNCTIONS CALLED:
NUMNOD

INPUT ARGUMENTS:
LABI--NAME OF NODE BEING DEFINED.
PRM1--MAXIMUM CAPACITY OF NODE
PRM2--NUMBER OF SERVERS

OUTPUT PARAMETERS:
NONE

INPUTS VIA COMMON:
IFATAL
NXTNOD
MAXNOD
LABEL2

OUTPUTS VIA COMMON:
IFATAL
NXTNOD
NODCAP
NODNAM
BALKS

WORKING VARIABLES:
NONE

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)

CHARACTER LABL*5

CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4.
& BALKS*8, DSTNAM*5

COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTHMAT(NONOD, NOMSG),
& CREATE(6, NOMSC), LASNOD(NOMSC), NODSTR(NOMSC),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSC, NONOD, 4), DSTXMT(NOMSC, NONOD, 4)

COMMON /COM2/ NODNAM(NONOD), DIST(NOMSC), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

HAS THIS NODE BEEN PREVIOUSLY DEFINED (AS INDICATED BY A
VALUE > 0)?

IF (NUMNOD(LABI) .GT. 0) THEN
  PRINT *, 'ERROR--NODE PREVIOUSLY DEFINED.'
  IFATAL = IFATAL + 1
  GO TO 999
ENDIF

INCREMENT NODE COUNT.
NXTNOD = NXTNOD + 1

VERIFY NODE COUNT STILL IN RANGE.
IF (NXTNOD .GT. MAXNOD) THEN
  PRINT *, 'ERROR--TOO MANY NODES DEFINED (IGNORED).'
  NXTNOD = MAXNOD
IFATAL = IFATAL + 1
NXTNOD = MAXNOD
GO TO 999
ENDIF
C SAVE NODE NAME.
NODNAM(NXTNOD) = LABL
C VERIFY VALIDITY OF MAXIMUM CAPACITY.
IF (PRM1 .LE. 0.0) THEN
   PRINT *, 'ERROR--MAXIMUM CAPACITY MUST BE GREATER THAN 0.'
   IFATAL = IFATAL + 1
ELSE
   NODCAP(1, NXTNOD) = PRM1
ENDIF
C NUMBER OF SERVERS MUST BE GREATER THAN 0.
IF (PRM2 .CT. 0.0) THEN
C SAVE THE NUMBER OF SERVERS.
   NODCAP(2, NXTNOD) = PRM2
ELSE
   PRINT *, 'ERROR--NUMBER OF SERVERS MUST BE GREATER THAN 0.'
   IFATAL = IFATAL + 1
ENDIF
C IS THIS NODE BLOCKED OR NO BALK NODE SPECIFIED?
IF (LABEL2 .EQ. 'BLOK' .OR. LABEL2 .EQ. ' ') THEN
C ITS BLOCKED OR NO BALK NODE SPECIFIED.
   BALKS(NXTNOD) = LABEL2
ELSE
C IS THE BALK NODE NAME VALID?
   IF (NUMNOD(LABEL2) .CT. 0) THEN
      ITS VALID--SAVE IT.
      BALKS(NXTNOD) = LABEL2
   ELSE
      ERROR--ITS NOT VALID.
      PRINT *, 'ERROR--ILLEGAL BALK NODE.'
      PRINT *, 'CORRECT THE NAME OR INSERT '
&       'CORRESPONDING NODE CARD.'
      IFATAL = IFATAL + 1
   ENDIF
ENDIF
999 RETURN
END
SUBROUTINE PATH
C
THIS SUBROUTINE HANDLES PATH CARDS BY SAVING THE INFORMATION
NECESSARY TO CONSTRUCT A PATH.
C
CALLED BY:
C MAIN
C SUBROUTINES CALLED:
C NONE
C FUNCTIONS CALLED:
C NUMNOD
C NUMTRM
C INPUT ARGUMENTS:
C NONE
C OUTPUT ARGUMENTS:
C NONE
C INPUTS VIA COMMON:
C IFATAL
C NXTMSG
C PTIIMAT
C LASNOD
C LABEL
C OUTPUTS VIA COMMON:
C IFATAL
C PTIIMAT
C LASNOD
C WORKING VARIABLES:
C NUMN--NODE NUMBER
C NUMT--TERMINATE NODE NUMBER
C
PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
& BALKS*8, DSTNAM*4
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXSC, NODCAP(2, NONOD), PTIIMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)
C
FIND NODE NUMBER FOR THIS LABEL.
NUMN = NUMNOD(LABEL)
C
FIND TERMINATE NODE NUMBER FOR THIS LABEL.
NUMT = NUMTRM(LABEL)
C
IF BOTH NUMN AND NUMT ARE 0, THEN THIS LABEL IS NEITHER A
C TERMINATE NOR A REGULAR NODE.
IF (NUMN .EQ. 0 .AND. NUMT .EQ. 0) THEN
    PRINT *, 'ERROR--UNDEFINED NODE.'
    PRINT *, 'CORRECT NODE NAME OR INSERT CORRESPONDING ',
    'NODE OR TERM CARD.'
    IFATAL = IFATAL + 1
ELSE
    VERIFY A PATH HAS NOT BEEN PREVIOUSLY DEFINED AT THIS LOCATION
    IF (PTIIMAT(LASNOD(NXTMSG), NXTMSG) .NE. 0.0) THEN
        PRINT *, 'ERROR--PREVIOUSLY DEFINED.'
        IFATAL = IFATAL + 1
PRINT *, 'ERROR--DUPLICATE PATH DEFINITION.'.
PRINT *, ' (A PATH FOR THIS MESSAGE AT THIS NODE WAS PREVIOUSLY SPECIFIED.)'
IFATAL = IFATAL + 1
ELSE
  IF (NUMN .GT. 0) THEN
    C A REGULAR NODE--MOVE THIS NODE NUMBER INTO THE PATH
    C MATRIX (PTHMAT).
    PTHMAT(LASNOD(NXTMSG), NXTMSG) = NUMN
    C THIS NODE NOW BECOMES THE PREVIOUS (LAST) NODE IN THIS
    C PATH.
    LASNOD(NXTMSG) = NUMN
  ELSE
    C A TERMINATE NODE--MOVE THE NEGATIVE OF THE TERMINATE NODE
    C NUMBER INTO THE PATH MATRIX (PTHMAT).
    PTHMAT(LASNOD(NXTMSG), NXTMSG) = -NUMT
    C THE TERMINATE NODE NOW BECOMES THE PREVIOUS (LAST) NODE
    C IN THIS PATH.
    LASNOD(NXTMSG) = -NUMT
  ENDIF
ENDIF
ENDIF
RETURN
END
SUBROUTINE PROC(LBL, PRM1, PRM2, PRM3)
C THIS SUBROUTINE HANDLES PROC CARDS BY SAVING THE INFORMATION
C NECESSARY FOR PROCESSING TIMES.
C CALLED BY:
C MAIN
C SUBROUTINES CALLED:
C NONE
C FUNCTIONS CALLED:
C NUNDST
C INPUT ARGUMENTS:
C LBL--PROCESSING TIME DISTRIBUTION NAME
C PRM1--1ST PARAMETER OF DISTRIBUTION
C PRM2--2ND PARAMETER OF DISTRIBUTION
C PRM3--3RD PARAMETER OF DISTRIBUTION
C OUTPUT ARGUMENTS:
C NONE
C INPUTS VIA COMMON:
C IFATAL
C NXTMSG
C LASNOD
C OUTPUTS VIA COMMON:
C IFATAL
C DSTPRC
C WORKING VARIABLES:
C NUM--DISTRIBUTION RATE NUMBER
C PARAMETER (NONOD=10, NOTRM=10, NOMSC=10)
CHARACTER LBL*5
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4.
& BALKS*8, DSTNAH*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAH(NONOD), DIST(NOMSC), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAH(9)
C FIND THE DISTRIBUTION NUMBER.
NUM = NUMDSTLBL
C IS DISTRIBUTION VALID (I.E., > 0)?
IF (NUM .EQ. 0) THEN
PRINT *, 'ERROR--ILLEGAL DISTRIBUTION SPECIFIED.'
IFATAL = IFATAL + 1
ELSE
C VALID, SAVE DISTRIBUTION NUMBER.
DSTPRC(NXTMSG, LASNOD(NXTMSG), 1) = NUM
C SAVE DISTRIBUTION PARAMETER 1.
DSTPRC(NXTMSG, LASNOD(NXTMSG), 2) = PRM1
C SAVE DISTRIBUTION PARAMETER 2.
DSTPRC(NXTMSG, LASNOD(NXTMSG), 3) = PRM2
C SAVE DISTRIBUTION PARAMETER 3.
DSTPRC(NXTMSG, LASNOD(NXTMSG), 4) = PRM3
40
ENDIF
RETURN
END
SUBROUTINE READ

THIS SUBROUTINE READS THE INPUT DATA DECK FROM FILE 20.

CALLED BY:
MAIN

SUBROUTINES CALLED:
NONE

FUNCTIONS CALLED:
NONE

INPUT ARGUMENTS:
NONE

OUTPUT ARGUMENTS:
NONE

INPUTS VIA COMMON:
IFATAL

OUTPUTS VIA COMMON:
IFATAL
PARAM1
PARAM2
PARAM3
PARAM4
PARAM5
CRDTYP
LABEL
LABEL2

WORKING VARIABLES:

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)

CHARACTER CARD67*67
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRNMAM*4.
CHARACTER BALKS*4, DSTNAM*5

COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
6 MAXMSG, NODCAP(2, NONOD), PTMAT(NONOD, NOMSG),
6 CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
6 PARM1, PARM2, PARM3, PARM4, PARM5,
6 DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRNMAM(NOTRM), CRDTYP,
6 LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

READ A CARD.
READ (10, 10, END=999) CRDTYP, CARD67
PRINT 30, CRDTYP, CARD67

RE-READ EACH CARD BASED UPON THE CARD TYPE TO OBTAIN THE NECESSARY
INFORMATION FOR THAT PARTICULAR CARD TYPE.
IF (CRDTYP .EQ. 'NODE') THEN
BACKSPACE 10
READ (10, 20, ERR=888) LABEL, PARM2, PARM3,
6 LABEL2
ELSE IF (CRDTYP .EQ. 'TERM' .OR. CRDTYP .EQ. 'PATH') THEN
BACKSPACE 10
READ (10, 40, ERR=888) LABEL
ELSE IF (CRDTYP .EQ. 'MESS') THEN
BACKSPACE 10

42
READ (10, 50, ERR=888) LABEL, PARM1, PARM2, PARM3, PARM4, PARM5, LABEL
ELSE IF (CRDTYP .EQ. 'PROC' .OR. CRDTYP .EQ. 'XMIT') THEN
  BACKSPACE 10
  READ (10, 70, ERR=888) LABEL, PARM3, PARM4, PARM1
ELSE IF (CRDTYP .EQ. 'BALK') THEN
  BACKSPACE 10
  READ (10, 90, ERR=888) LABEL, PARM1
ELSE IF (CRDTYP .NE. 'END ') THEN
  PRINT 100
  IFATAL = IFATAL + 1
ENDIF
GO TO 1000
888 PRINT 110
  CRDTYP = 'ERR'
  IFATAL = IFATAL + 1
1000 RETURN
999 PRINT 120
  STOP
10 FORMAT (A4, A67)
20 FORMAT (5X, A5, 2F10.5, A4)
30 FORMAT (1X,A4, A67)
40 FORMAT (5X, A5)
50 FORMAT (5X, A5, 5F10.5, A4)
70 FORMAT (5X, A5, 3F10.5)
90 FORMAT (5X, A5, F10.5)
100 FORMAT (' ERROR--INVALID CARD TYPE. ')
110 FORMAT (' ERROR--CONVERSION ERRORS OCCURRED.', /,
  & 7X, 'CHECK INPUT FORMATS. ')
120 FORMAT (' UNEXPECTED END-OF-FILE. JOB TERMINATED. ')
END
SUBROUTINE TERM (LABL)

THIS SUBROUTINE HANDLES TERM CARDS BY SAVING THE TERMINATE NODE NAME.

CALLED BY:
   MAIN
SUBROUTINES CALLED:
   NONE
FUNCTIONS CALLED:
   NUMTRM
INPUT ARGUMENTS:
   LABL--TERMINATE NODE NAME
OUTPUT ARGUMENTS:
   NONE
INPUTS VIA COMMON:
   IFATAL
   NXTTRM
   MAXTRM
OUTPUTS VIA COMMON:
   IFATAL
   NXTTRM
   TRMNAM
WORKING VARIABLES:
   NONE

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER LABL*5
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIHMT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NOTRM(NOTRM), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

HAS THIS TERMINATE NODE BEEN PREVIOUSLY DEFINED?
IF (NUMTRM(LABL) .GT. 0) THEN
   PRINT *, 'ERROR--TERM PREVIOUSLY DEFINED.'
   IFATAL = IFATAL + 1
ELSE
   INCREMENT TERMINATE NODE COUNT.
   NXTTRM = NXTTRM + 1
   IS COUNT STILL IN RANGE?
   IF (NXTTRM .GT. MAXTRM) THEN
      PRINT *, 'ERROR--TOO MANY TERMS DEFINED (IGNORED).'
      IFATAL = IFATAL + 1
      NXTTRM = MAXTRM
   ELSE
      SAVE THIS TERMINATE NODE NAME.
      TRMNAM(NXTTRM) = LABL
   ENDF
ENDDF
SUBROUTINE XMIT (LBL, PRM1, PRM2, PRM3)

This subroutine handles XMIT cards by saving the information necessary for transmit time.

Called by: 
MAIN

Subroutines called: 
NONE

Functions called: 
NUMDST

Input arguments: 
LBL -- Transmit time distribution name
PRM1 -- 1st parameter of distribution
PRM2 -- 2nd parameter of distribution
PRM3 -- 3rd parameter of distribution

Output arguments: 
NONE

Inputs via common: 
IFATAL
NXTMSG
LASNOD

Outputs via common: 
IFATAL
DSTXMT

Working variables: 

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER LBL*5
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRNMAM*4,
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTNUMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DISTPRC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRNMAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

Find the distribution number.
NUM = NUMDST(LBL)

Is distribution number valid?
IF (NUM .EQ. 0) THEN
  PRINT *, 'ERROR--ILLEGAL DISTRIBUTION SPECIFIED.'
  IFATAL = IFATAL + 1
ELSE
  Save the distribution number.
  DSTXMT(NXTMSG, LASNOD(NXTMSG), 1) = NUM
  Save parameter 1 of the distribution.
  DSTXMT(NXTMSG, LASNOD(NXTMSG), 2) = PRM1
  Save parameter 2 of the distribution.
  DSTXMT(NXTMSG, LASNOD(NXTMSG), 3) = PRM2
  Save parameter 3 of the distribution.
  DSTXMT(NXTMSG, LASNOD(NXTMSG), 4) = PRM3

46
FUNCTION NUMDIST(NAME)

   THIS FUNCTION RETURNS THE DISTRIBUTION NUMBER FOR NAME (IF
   A VALID DISTRIBUTION NAME) OR 0 (IF INVALID).

   CALLED BY:
   BLIDOUT
   MCSS
   PROC
   X'1T

   SUBROUTINES CALLED:
   NONE

   FUNCTIONS CALLED:
   NONE

   INPUT ARGUMENTS:
   NAME--DISTRIBUTION NAME

   OUTPUT ARGUMENTS:
   NONE

   INPUTS VIA COMMON:
   DSTNAM

   OUTPUTS VIA COMMON:
   NONE

   WORKING VARIABLES:
   I--DO LOOP COUNTER

   PARAMETER (NONOD=10, NOTRM=10, NMSG=10)
   CHARACTER NAME*5
   CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
   & BALKS*4, DSTNAM*5
   COMMON /COM1/ IFATAL, Extend, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
   & MAXMSG, NODCAP(2, NONOD), PTHMAT(NONOD, NMSG),
   & CREATE(6, NMSG), LASNOD(NOMSC), NODSTR(NOMSC),
   & PARM1, PARM2, PARM3, PARM4, PARM5,
   & DSTPBC(NOMSG, NONOD), DSTXMT(NOMSG, NONOD, 4)
   COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
   & LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

   ESTABLISH DEFAULT RETURN VALUE.
   NUMDIST = 0
   DO 10 I = 1, 9

   IF FOUND, RETURN 1.
   IF (NAME .EQ. DSTNAM(I)) THEN
      NUMDIST = I
   GO TO 20
   ENDF

   CONTINUE

10   CONTINUE

20   RETURN

END
FUNCTION NUMNOD(NODE)

C THIS FUNCTION RETURNS THE NODE NUMBER FOR NODE (IF THE NODE HAS
C BEEN PREVIOUSLY DEFINED) OR 0 (IF UNDEFINED).
C
C CALLED BY:
C MAIN
C CREATE
C PATH
C
C SUBROUTINES CALLED:
C NONE
C
C FUNCTIONS CALLED:
C NONE
C
C INPUT ARGUMENTS:
C NODE--NODE NAME
C
C OUTPUT ARGUMENTS:
C NONE
C
C INPUTS VIA COMMON:
C MAXNOD
C NODNAM
C
C OUTPUTS VIA COMMON:
C NONE
C
C WORKING VARIABLES:
C I--DO LOOP COUNTER

C PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)
CHARACTER NODE*4
CHARACTER CRDYP*4, LABEL*4, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*4
& BALKS*4, DSTNAM*5
COMMON /COM1/ IFATAL, NXTNOD, NXTTRM, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTMMAT(NONOD, NOMSG),
& CREATE(6, NOMSG), LASNOD(NOMSG), NODSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRTC(NOMSG, NONOD, 4), DSTXMT(NOMSG, NONOD, 4)
COMMON /COM2/ NODNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)
C
C ESTABLISH DEFAULT RETURN VALUE.
NUMNOD = 0
DO 10, I = 1, MAXNOD
C IF FOUND, RETURN I.
C IF (NODE .EQ. NODNAM(I)) THEN
   NUMNOD = I
   GO TO 20
ENDIF
10 CONTINUE
20 RETURN
END
FUNCTION NUMTRM (NODE)

THIS FUNCTION RETURNS THE TERMINATE NODE NUMBER FOR NODE (IF THE TERMINATE NODE HAS BEEN DEFINED) OR 0 (IF UNDEFINED).

CALLED BY:
MAIN
PATH
TERM

SUBROUTINES CALLED:
NONE

FUNCTIONS CALLED:
NONE

INPUT ARGUMENTS:
NODE--TERMINATE NODE NAME

OUTPUT ARGUMENTS:
NONE

INPUTS VIA COMMON:
MAXTRM
TRMNAM

OUTPUTS VIA COMMON:
NONE

WORKING VARIABLES:
I--DO LOOP COUNTER

PARAMETER (NONOD=10, NOTRM=10, NOMSG=10)

CHARACTER NODE*4
CHARACTER CRDTYP*4, LABEL*5, LABEL2*4, NODNAM*4, DIST*5, TRMNAM*
& BALKS*4, DSTNAM*5

COMMON /COM1/ IFATAL, NXTNOD, NXTTRH, NXTMSG, MAXNOD, MAXTRM,
& MAXMSG, NODCAP(2, NONOD), PTIMAT(NONOD, NOMSG),
& CREATE(6, NOMSC), LASNOD(NOMSG), NONSTR(NOMSG),
& PARM1, PARM2, PARM3, PARM4, PARM5,
& DSTPRC(NOMSC, NONOD, 4), DSTXMT(NOMSC, NONOD, 4)
COMMON /COM2/ NONNAM(NONOD), DIST(NOMSG), TRMNAM(NOTRM), CRDTYP,
& LABEL, LABEL2, BALKS(NONOD), DSTNAM(9)

ESTABLISH DEFAULT RETURN VALUE.
NUMTRM = 0
DO 10 I = 1, MAXTRM
IF FOUND, RETURN I.
IF (NODE .EQ. TRMNAM(I)) THEN
NUMTRM = I
GO TO 20
ENDIF
10 CONTINUE
20 RETURN
END

50
Appendix B: Operating Instructions

Introduction
The SLAM Pre-Processor is designed to operate in a batch mode, with card or card image inputs from file 10. Eight legal card types describe the network to be simulated, and the flow of messages through the network.

Card Types
The eight legal card types are BALK, END, MESS, NODE, PATH, PROC, TERM, and XMIT. Each of these are described in more detail below. Samples of each card type may be found in Appendix E.

BALK Card. When the capacity of a node has been exceeded, a balk node may be specified (more details concerning balking may be found in the description of the NODE card). A BALK card is used to define the beginning of a balk route specification (the route to be taken should a node’s capacity be exceeded) by specifying the name of the node for which a balk route is being provided, and the message number for which the route applies.

END Card. An END card marks the end of the SLAMPP input deck.

MESS Card. A MESS card marks the beginning of a message description and defines the message arrival rate, the time of the first message arrival, the number of message releases (number of messages to be created), and the name of the first node on the message path.

NODE Card. A NODE card defines a node (by name), its maximum capacity, the number of servers (the number of messages which may be
processed at the same time) at this node, and balking or blocking information.

When a message arrives at a node whose queue is at its maximum capacity, one of three actions may occur: the message may balk, the message may block, or the message may be lost. If balking is specified, the message will go to an alternate (balk) node. If blocking is specified, the message "backs up" in the network prior to the node and waits until a message leaves the node, at which time it moves into the node. If neither balking nor blocking is specified, the message will simply be lost.

**PATH Card.** A PATH card specifies the name of the node which is next in the path.

**PROC Card.** A PROC card describes the amount of time a message takes to be processed prior to leaving a node.

**TERM Card.** A TERM card defines (by name) those nodes at which a message may leave the network, called "terminate nodes."

**XMIT Card.** An XMIT card describes the amount of time it takes to transmit a message to the next node on the path after a message has been processed.

**Distribution Rates**

Three card types (MESS, PROC, and XMIT) allow the user to specify a time duration. In the case of the MESS card, the time is the time between message creations while, for PROC and XMIT, it is the time for processing and transmitting a message, respectively.

SLAMPP will recognize nine distributions as valid. Each
distribution requires from one to three parameters, which should be input in the order given. These are described below (Pritsker, 1979: 528-534).

**BETA.** Beta distribution with two parameters: Theta and phi.

**CONST.** Constant duration with one parameter: length.

**EXPON.** Exponential distribution with one parameter: mean.

**GAMA.** Gamma distribution with two parameters: Beta and alpha.

**NPSSN.** Poisson distribution with one parameter: mean.

**RLOGN.** Lognormal distribution with two parameters: mean and standard deviation.

**RNORM.** Normal distribution with two parameters: mean and standard deviation.

**TRIAG.** Triangular distribution with three parameters: interval low value, mode, and interval high value.

**UNFRM.** Uniform distribution with two parameters: interval low value and interval high value.

**Input Formats**

Each input card consists of a card type, plus (depending upon card type) up to seven additional data areas. Regardless of card type, each data area (or field) is located in the same card columns. Field one is in columns 1-4 and field two in 6-10, while the remaining fields begin in columns 11, 21, 31, 41, 51, and 61, and are each 10 columns wide.

Two types of data may be placed within each field: alphanumerict or numeric. Alphanumeric data must be left justified within the field. All numeric data is input as real values, and may fall any place within
a field; however, a decimal point must be explicitly included with the number. Thus, the number "1" should be entered as "1." or "1.0". Only the first five decimal places are significant.

Although each numeric field should be entered as a real value (to keep all data field formats consistent for all card types), some values are internally converted to integers by truncation. Therefore, if the data item is listed as "integer," ensure that no decimal places are included.

The format for each of the eight card types is shown below. The number within parenthesis is the field number associated with that data item.

**BALK.** (1) Card type keyword "BALK". (2) Name of the node (not a terminate node) for which a balk route is being specified. This node name should have appeared as a balk node on a NODE card. (3) The integer message number for which a balk route is being specified. Message numbers are assigned sequentially (beginning with one) in the order found in the SLAMPP input deck (see the section on message numbering later).

**END.** (1) Card type keyword "END".

**NESS.** (1) Card type keyword "NESS". (2) Distribution name for the message creation rate. (3) Parameter one of the distribution. (4) Parameter two of the distribution (if only one parameter is required, enter zero). (5) Parameter three of the distribution (if only one or two parameters are required, enter zero). (6) The integer time of creation of the first message. (7) The integer number of messages to be created. (8) The name of the first node in this message's path.
NODE. (1) Card type keyword "NODE". (2) The name of the node being defined (four or fewer characters). (3) The integer maximum capacity of this node (the maximum number of messages which may be waiting at this node at any one time). (4) The integer number of servers at this node (the maximum number of messages which may be processed concurrently). (5) Balking or blocking information. If balking is to occur, place the name of the node to which messages should balk if maximum capacity is exceeded. If blocking is to occur, place the keyword "BLOK" into this field. If neither balking or blocking is desired (meaning that an arriving message will be lost if maximum capacity has been reached), leave this field blank.

PATH. (1) Card type keyword "PATH". (2) The name of the next node along this message or balk route's path.

PROC. (1) Card type keyword "PROC". (2) The distribution name for the processing time at this node. (3) Parameter one of the distribution. (4) Parameter two of the distribution (if only one parameter is required, enter zero). (5) Parameter three of the distribution (if only one or two parameters are required, enter zero).

TERM. (1) Card type keyword "TERM". (2) The name (four or fewer characters) of the terminated node being declared.

XMIT. (1) Card type keyword "XMIT". (2) The distribution name for the transmission time at this node. (3) Parameter one of the distribution. (4) Parameter two of the distribution (if only one parameter is required, enter zero). (5) Parameter three of the distribution (if only one or two parameters are required, enter zero).
Building the Input Deck

The input deck can be divided into four categories, which are described below. The order of these descriptions is the same as the order in which these categories must fall within the SLAMPP input deck.

Declarations. Before any message can be defined, all nodes within the network must be defined. Each node must be described and defined (using NODE cards). Following the node declarations comes the declaration of all terminate nodes (using TERM cards).

Message Descriptions. Following the declaration portion of the SLAMPP input is the message description area. A message description may involve up to four card types: MESS, PROC, XMIT and PATH.

The first card in the description of any message is a MESS card specifying the creation rate for this message and the first node along the path. In essence, this card specifies when and how often this message will arrive, and the name of the node where the message will first arrive.

While at a node, a message may be processed in some manner which requires time. This processing time can be expressed by using a PROC card. If processing is not required, this card may be omitted.

After all processing is complete, this message is ready to be transmitted to the next node along the path. This transmission may require time, and can be defined using an XMIT card. If no transmission time is needed, this card may be omitted.

The next node along this message's path (that is, the node to which the message is being transmitted) is defined by a PATH card. This PROC, PATH and XMIT pattern is repeated until the entire path of the message
has been described. The last node of every message's path must be a terminate node.

Subsequent messages are described by inserting another MESS card and following the above instructions again.

Balk Route Descriptions. One of the items specified on a NODE card is whether balking will occur or not. If balking is specified, there is a possibility that every message that uses this node could be redirected to the balk node. Therefore, a path must exist for every message which could possibly enter a balk node. If a path already exists for a message at the balk node from the message's original description, nothing more need be done for this message. However, if a path does not exist, one must be supplied by means of a BALK card. All balk route descriptions (if any are present) follow the last message description.

The BALK card specifies the balk node name and the message number for which a balk route is being provided. Following the BALK card, PROC, XMIT and PATH cards are repeated (just as in a message description) until the entire balk path is defined. As with a regular message path, balk paths must end at a terminate node.

Additional balk routes may be supplied by inserting another BALK card and repeating the above steps.

End of Deck. The last card of the input deck must be an END card. No additional data will be read beyond the END card.

Design Considerations

Designing the SLAMPP Network. Experience has shown that even the most simple network can become confusing unless a systematic method is
used to translate that network into SLAHP inputs. In order to minimize the number of input errors, a few simple guidelines are suggested.

First, determine exactly which nodes and terminate nodes will be required for the network. In the case of nodes, also determine the number of servers, maximum capacity, and balking or blocking characteristics. Translate this information into the appropriate NODE or TERM cards.

Second, determine exactly which messages are to be included. A good method to follow is to "outline" the message directly into SLAHP inputs. That is, code the appropriate MESS card for the message, and follow this with the applicable PROC, XMIT and PATH cards to describe the message. After the first message is completely defined, begin defining the second, then the third, etc., until all messages have been defined.

Third, define any balk routes which may be required. To do this, examine each node, in succession, which has a balk node specified on its NODE card. Then, for every message which uses this node, ensure a route exists for this message from the balk node. This route may already exist if the message originally was routed using this balk node as part of its path. For example, as might occur if a message went from N1 to N2 to N3 to N4, and N1 balked to N3. In this case, if N1 was at maximum capacity the message would skip N1 and N2 and go directly to N3. If a route does not already exist for this message, then a route must be defined by using a BALK card. As with a message, follow the BALK card with the necessary PROC, XMIT and PATH cards to describe the route.

Fourth, make sure that any routes added in step three that use a
node which can balk has its own balk routes specified. Repeat this step until all routes and all necessary balk routes are specified.

Lastly, supply an EHD card to the input deck.

Message Numbering. Balk routes are specified using the number of the message for which a route is being supplied. It is, therefore, important to know how messages are numbered.

Messages are numbered consecutively (beginning at one) in the order they appear in the input deck. Thus, the first message found is message number one; the second, number two, etc.

When additional messages are being added to the network, always add the new message(s) after the end of the last message of the network. In this manner, any balk routes which may have been previously defined will not need any changes. Otherwise, the BALK cards will need changing to reflect the new message numbers.

Balk Route Considerations. Balk route design provides the user with several methods which may be used. To familiarize the user with these methods, they are described below.

First, no balk route may be necessary and therefore, the user need not do anything. However, this is only applicable to nodes which do not have a balk node specified.

Second, the balk route may be an entirely different route from the path defined for the message (by the HESS card). The primary consideration here is to ensure that this balk route ends on a terminate node.

Third, the balk node may have been included within the message's original path. In this case, a route already exists for this message.
from the balk node, and the user need not (in fact, cannot) provide another route.

Fourth, the balk route may merge back into the message route. This means that a portion of the balk route is entirely different from the message path; however, at some point in the balk route a node defined in the message path is used. From this node forward, the balk route is the same as the message path. To illustrate this, assume a message has path N1, N2, N4, N3 and N5 (in that order). If N2 balks to N7, then a merging balk route might be N7, N8 and N4. In this example, a message would balk from N2 to N7 and then traverse N8 and N4. The remainder of the path from N4 would be the same as the message path. In essence, the balk route would be N7, N8, N4, N3 and N5. Diagramatically, this could be viewed as:

![Diagram showing the balk and message routes]

N1 → N2 → N7 → N8 → N4 → N3 → N5
All messages which arrive at a node are required to balk to the same balk node. Occasionally, however, different balk nodes may be desired for each different message. This can be accomplished by using a "dummy" balk node whose only function is to redirect messages to their appropriate nodes. The recommended procedure is to define this dummy balk node as having a very large capacity (9999) and a large number of servers (9999). Since this node does not represent a real node within the network, following this definition prevents this node from becoming a "bottleneck" in the system and affecting statistics. Immediately following the BALK cards for the messages to be redirected, insert a PATH card to the desired node. This causes the message to balk to the dummy node and then be transmitted to the appropriate node. However, since both balking and transmitting (in this case) use zero time, the net result is that the message goes directly to its appropriate node.
Appendix C: Input Errors

The validity of the SLAM Pre-Processor input is verified at several points. Should any errors be detected, an error message is printed and the SLAM input source code is not created. Below is a list of the error messages which may be produced, along with an explanation and a possible cause or correction.

CONVERSION ERRORS OCCURRED. A conversion error occurred trying to read the input file. The most likely cause is failure to follow the correct input format for the card type involved. Correct the format.

DUPLICATE PATH DEFINITION. An attempt was made to define a path for a particular message using a node already defined on that message's path. Frequently caused by trying to define a balk path for a message which overlaps the message's path.

END CARD EXPECTED. Additional input data followed the end of the last balk route or, if no balk routes are specified, the end of the last message. Either this data is incorrect (and should be removed), or a problem exists in the last message or balk route to be specified.

INCOMPLETE BALK PATH FOR MESSAGE n FROM NODE name. The balk path for message number n (which balks from node name) does not end on a terminate node. Either the path is incorrect (and should be corrected) or the balk route was not supplied as required.

ILLEGAL BALK NODE. The balk node name specified was not defined in a NODE statement. Either supply a corresponding NODE statement, or correct the node name.

ILLEGAL BALK NODE NAME. The balk node name specified on the BALK
card is incorrect and should be corrected.

ILLEGAL DISTRIBUTION SPECIFIED. The distribution name specified is not valid in the SLAMPP (see Appendix B). Correct to a valid name.

INVALID BALK NODE (SKIPPING TO NEXT BALK). The balk node name is incorrect and should be corrected. The remainder of this balk path is ignored.

INVALID CARD TYPE. The card type specified is not a valid SLAMPP card type (see Appendix B). Correct the card type or remove from the input deck.

INVALID MESSAGE NUMBER (SKIPPING TO NEXT BALK). The message number specified on the BALK card is invalid (either zero, negative, or greater than the number of messages input to SLAMPP). Correct the number. The remainder of this balk path is ignored.

MAXIMUM CAPACITY MUST BE GREATER THAN 0. The maximum capacity specified for this node was either zero or negative. Correct to a positive value.

MESS CARD MUST FOLLOW TERM CARDS. A card type other than MESS was found following the last TERM card. The input deck is out of order (and should be reordered).

MESSAGE MUST END ON A TERMINATE NODE. The message path being defined did not end on a terminate node, leaving this path incomplete. Complete the path.

MISSING BALK PATH CARD. SKIPPING TO NEXT BALK. The path for this balk route did not contain a PATH card. The remainder of this path is ignored.

MISSING PATH CARD. SKIPPING TO NEXT MESS. The path for this
message route did not contain a PATH card. The remainder of this message is ignored.

NODE CARDS MUST BE FIRST. The first card type found was not a NODE card. Inputs are out of order and should be reordered.

NODE \texttt{name} IS NOT USED IN A PATH. The node \texttt{name}, which was defined in a NODE card, was not used in any path. Either place \texttt{name} in a path, or remove the corresponding NODE card.

NODE PREVIOUSLY DEFINED. The node being defined by this NODE card has appeared previously on a NODE card. Remove one of the NODE cards.

NUMBER OF SERVERS MUST BE GREATER THAN 0. The number of servers specified for this node was either zero or negative. Correct to a positive value.

TERM CARDS MUST FOLLOW NODE CARDS. The first card type found following the last NODE card was not a TERM card. Inputs are out of order and should be reordered.

TERM PREVIOUSLY DEFINED. The terminate node being defined by this TERM card has appeared previously on a TERM card. Remove one of the TERM cards.

THE BALK PATH FOR MESSAGE \texttt{n} CONTAINS A CYCLE FROM NODE \texttt{name}. The balk route specified for message number \texttt{n} from node \texttt{name} loops back upon itself. Correct the path to remove the cycle.

TOO MANY MESSAGES DEFINED. The number of messages input to the SLAMPP exceeds the program capacity. Either remove the excess messages, or recompile the Pre-Processor with an increased value for NOMSG (see Chapter III).

TOO MANY NODES DEFINED (IGNORED). The number of nodes input to the
SLAIPP exceeds the program capacity. This node declaration is ignored. Either remove the excess nodes, or recompile the Pre-Processor with an increased value for NONOD (see Chapter III).

TOO MANY TERMS DEFINED (IGNORED). The number of terminate nodes input to the SLAIPP exceeds the program capacity. This terminate node declaration is ignored. Either remove the excess terminate nodes, or recompile the Pre-Processor with an increased value for NOTRM (see Chapter III).

UNDEFINED NODE. The node specified was not declared in a NODE or TERM statement. Insert the corresponding declaration.

UNEXPECTED END-OF-FILE. JOB TERMINATED. The end of the input data was encountered when SLAIPP expected additional data, usually caused by a missing END card or previous errors. Insert an END card or correct the other errors.
Appendix D: Interpreting the SLAM Output

Introduction

Upon successful completion of a network simulation, the user must be able to interpret the SLAM output. Summarized here is a description of the outputs. Refer to the sample SLAM output at the end of this appendix for examples.

SLAM Output

The output of interest to the user begins on the page labelled "SLAM SUMMARY REPORT." The first item to notice is the "Current Time." The time shown is the time the last message left the network, leaving the network empty.

Terminate Nodes

Following this is the "Statistics for Variables Based on Observations." Statistics are produced for each terminate node within the network and are identified by terminate node name. The statistics are based upon the time interval between message arrivals at the terminate node. The mean, standard deviation, and coefficient of variation of the time between message arrivals are listed. Also, the minimum and maximum arrival intervals and the number of messages to arrive at this terminate node are shown.

Queues

File statistics are printed next. These statistics deal with the
queues for each of the nodes in the network. The queue statistics are listed in the same order as their NODE cards in the SLAHP network. An easier way, perhaps, to identify the node's name is to look at the next section of the SLAM output titled "Resource Statistics." The "Resource Number" for a particular node name (listed under "Resource Label") is the same as that node's "File Number." Note that the last file number listed is a SLAM work file and should be ignored.

The statistics shown include the average length, standard deviation and maximum length of each queue. The average time a message waited to be processed at a queue is also presented. Note that messages may have been lost at nodes which do not have balking nor blocking specified if the maximum length of that node's queue is the same as that node's maximum capacity.

Service and Transmission

The "Resource Statistics" represents statistics dealing with the service and transmission time at each node. The first group of data represents node utilization while the second group represents node availability.

In the utilization group, "Current Capacity" is the number of servers available at this node. The "Average Utilization" is the average number of servers to be used. The maximum number of servers used concurrently and the current number of servers being used (at end of simulation) are also shown.

In the available group, the average number of servers available, plus the minimum and maximum number available are shown.
**Simulation Project SLAM**

**BY SLAMPP**

**Date 9/19/1982**

**Run Number 1 of 1**

**Current Time** .1235E+04

Statistical arrays cleared at time 0.

### **Statistics for Variables Based on Observation**

<table>
<thead>
<tr>
<th>EXIT INTERVAL</th>
<th>TRM1</th>
<th>.563E+02</th>
<th>.158E+03</th>
<th>.280E+01</th>
<th>.172E+01</th>
<th>.113E+04</th>
<th>295</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT INTERVAL</td>
<td>TRM2</td>
<td>.381E+03</td>
<td>.356E+03</td>
<td>.934E+00</td>
<td>.903E+01</td>
<td>.115E+04</td>
<td>74</td>
</tr>
<tr>
<td>EXIT INTERVAL</td>
<td>TRM3</td>
<td>.119E+03</td>
<td>.131E+03</td>
<td>.110E+01</td>
<td>.938E+01</td>
<td>.764E+03</td>
<td>238</td>
</tr>
</tbody>
</table>

### **File Statistics**

<table>
<thead>
<tr>
<th>FILE NUMBER</th>
<th>ASSOCIATED NODE</th>
<th>AVERAGE LENGTH</th>
<th>STANDARD DEVIATION</th>
<th>MAXIMUM LENGTH</th>
<th>CURRENT AVERAGE LENGTH</th>
<th>WAIT TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AWAIT</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>AWAIT</td>
<td>0.001</td>
<td>0.045</td>
<td>3</td>
<td>0</td>
<td>0.113</td>
</tr>
<tr>
<td>3</td>
<td>AWAIT</td>
<td>2.991</td>
<td>10.750</td>
<td>61</td>
<td>0</td>
<td>92.125</td>
</tr>
<tr>
<td>4</td>
<td>AWAIT</td>
<td>9.904</td>
<td>17.473</td>
<td>50</td>
<td>0</td>
<td>152.550</td>
</tr>
<tr>
<td>5</td>
<td>AWAIT</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>AWAIT</td>
<td>4.399</td>
<td>12.353</td>
<td>50</td>
<td>0</td>
<td>28.399</td>
</tr>
<tr>
<td>7</td>
<td>AWAIT</td>
<td>1.181</td>
<td>1.249</td>
<td>10</td>
<td>0</td>
<td>6.967</td>
</tr>
<tr>
<td>8</td>
<td>AWAIT</td>
<td>9.869</td>
<td>13.005</td>
<td>30</td>
<td>0</td>
<td>570.047</td>
</tr>
<tr>
<td>9</td>
<td>AWAIT</td>
<td>0.628</td>
<td>2.909</td>
<td>15</td>
<td>0</td>
<td>3.790</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>4.118</td>
<td>3.395</td>
<td>19</td>
<td>0</td>
<td>1.162</td>
</tr>
</tbody>
</table>

### **Resource Statistics**

<table>
<thead>
<tr>
<th>RESOURCE NUMBER</th>
<th>RESOURCE LABEL</th>
<th>CURRENT CAPACITY</th>
<th>AVERAGE UTIL</th>
<th>STANDARD DEVIATION</th>
<th>MAXIMUM UTIL</th>
<th>CURRENT UTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALFA</td>
<td>1</td>
<td>.18</td>
<td>.381</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>BLKI</td>
<td>1</td>
<td>.00</td>
<td>.052</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>BLK2</td>
<td>1</td>
<td>.10</td>
<td>.298</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>BETA</td>
<td>1</td>
<td>.29</td>
<td>.451</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>DLTA</td>
<td>5</td>
<td>.05</td>
<td>.294</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>RESOURCE NUMBER</td>
<td>RESOURCE LABEL</td>
<td>CURRENT AVAILABLE</td>
<td>AVERAGE AVAILABLE</td>
<td>MINIMUM AVAILABLE</td>
<td>MAXIMUM AVAILABLE</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>P1</td>
<td>1</td>
<td>0.46</td>
<td>0.498</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>RHO</td>
<td>1</td>
<td>0.03</td>
<td>0.179</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>ZETA</td>
<td>1</td>
<td>0.39</td>
<td>0.489</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>GAMMA</td>
<td>2</td>
<td>0.17</td>
<td>0.481</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Resource labels (ALFA, BLK1, BLK2, BETA, DLTA, P1, RHO, ZETA, GAMMA) are listed along with their corresponding current available, average available, minimum available, and maximum available values.
Appendix E: Test and Evaluation

In order to check the validity of the SLAMPP, a test case was designed and evaluated. This test case was designed to use at least once every capability of SLAMPP. Specifically, nodes with balking, nodes with blocking and nodes with neither were used. Each of the nine valid distribution types found in Appendix B was used in each of their legal positions, that is, on the message creation card (MESS), process card (PROC) and the transmit card (XMIT). The default process and transmit time of zero was also demonstrated. Since balking was specified on two nodes, balk route descriptions were required in addition to regular route descriptions.

This test case was developed using the design considerations found in Appendix B. All of the nodes to be included were first determined and translated into NODE and TERM cards (a listing of the SLAMPP inputs may be found later in this appendix). The messages to be included were determined and placed into the SLAMPP input deck. Lastly, all balk routes which were necessary were supplied. The message creations and routings are diagrammatically illustrated on the following pages.

After all SLAMPP inputs were completed, the SLAMPP program was executed to produce the SLAM source code, a copy of which is at the end of this appendix. This source code was manually inspected to ensure that the source was an accurate representation of the network to be modeled. After acceptance, the source code was executed by SLAM to verify that SLAM would indeed accept the code. The SLAM execution listing is also included in this appendix.
After repeated executions of SLAIPP, no errors could be detected and the program was accepted.
MESSAGE CREATIONS

Msg 1 (EXPON) → Beta

Msg 2 (NPSSN) → Beta

Msg 3 (BETA) → Rho

Msg 4 (GAMA) → Delta

Msg 5 (RLOGN) → Gamma

Msg 6 (RNORM) → Zeta

Msg 7 (UNFRM) → Gamma

Msg 8 (TRIAG) → Beta

Msg 9 (CONST) → Gamma
**SLAM PRE-PROCESSOR**

**SUMMARY OF PROGRAM INPUTS:**

<table>
<thead>
<tr>
<th>NODE</th>
<th>ALFA</th>
<th>50.0</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE</td>
<td>BLK1</td>
<td>9999.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>BLK2</td>
<td>9999.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>BETA</td>
<td>50.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>DLTA</td>
<td>20.0</td>
<td>5.0</td>
</tr>
<tr>
<td>NODE</td>
<td>PI</td>
<td>50.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>RHO</td>
<td>10.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>ZETA</td>
<td>30.0</td>
<td>1.0</td>
</tr>
<tr>
<td>NODE</td>
<td>GAMA</td>
<td>15.0</td>
<td>2.0</td>
</tr>
<tr>
<td>TERM</td>
<td>TRM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERM</td>
<td>TRM2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERM</td>
<td>TRM3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESS</td>
<td>EXPON2.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>PROC</td>
<td>TRIAG2.0</td>
<td>7.0</td>
<td>9.0</td>
</tr>
<tr>
<td>PATH</td>
<td>DLTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>CONST1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>PI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC</td>
<td>EXPON0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>BETA</td>
<td>3.12</td>
<td>6.14</td>
</tr>
<tr>
<td>PATH</td>
<td>TRM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESS</td>
<td>NPSSN1.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>PATH</td>
<td>DLTA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>CONST0.5</td>
<td>.95</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>PI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>RLOGN1.333</td>
<td>.888</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>RHO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC</td>
<td>EXPON.23175</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>NPSSN.103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>ZETA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC</td>
<td>RNORM2.1</td>
<td>.44</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>GAMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC</td>
<td>CONST2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>CONST1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>TRM1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESS</td>
<td>BETA</td>
<td>2.70</td>
<td>1.0</td>
</tr>
<tr>
<td>PROC</td>
<td>UNFRM2.1</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>PI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>CONST6.55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>TRM2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESS</td>
<td>GAMA</td>
<td>6.2</td>
<td>5.1</td>
</tr>
<tr>
<td>PROC</td>
<td>CONST9.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>NPSSN0.273</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>GAMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>ALFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROC</td>
<td>CONST3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMIT</td>
<td>RLOGN2.7</td>
<td>.334</td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>TRM3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| MESS | PROC | CONST | XMIT | TRIAC | PATH | PI | PROC | CONST | XMIT | TRIAC | PATH | TRI1 | MESS | RNSRM | 4.1 | PATH | BETA | PROC | NPSN | 0.667 | XMIT | NPSN | 0.669 | PATH | RHD | XMIT | GAMA | 0.45 | | PATH | DLTA | PROC | RNORM | 0.382 | XMIT | RLOGN | 0.543 | PATH | TRM3 | MESS | RNSRM | 0.15 | PATH | ZETA | PROC | RNORM | 0.25 | XMIT | CONST | 0.0 | PATH | TRM3 | MESS | TRIAC | 2.0 | XMIT | TRIAC | 3.0 | PATH | TRM3 | MESS | CONST | 1.5 | PROC | UNFRM | 0.2 | XMIT | CONST | 0.621 | PATH | PI | PROC | BETA | 0.5 | PATH | BETA | PATH | TRI1 | BALK | BLK1 | 1.0 | XMIT | Const | 0.5 | PATH | RHO | PROC | NPSN | 1.6 | XMIT | EXPON | 0.002 | PATH | TRM1 | BALK | BLK1 | 2.0 | PATH | DLTA | BALK | BLK2 | 2.0 | XMIT | GAMA | 1.234 | 2.345 | PATH | BETA | BALK | BLK1 | 3.0 | PROC | RNORM | 0.45 | XMIT | CONST | 2.678 | PATH | BETA | PATH | DLTA | PROC | RLOGN | 1.2 | PATH | ZETA | XMIT | BETA | 6.1 | PATH | TRM2 | BALK | BLK1 | 4.0 | XMIT | UNFR | 0.09 | 2.1 | 78
NO ERRORS DETECTED. SLAM SOURCE WILL BE CONSTRUCTED.
1  GEN, SLAMPP, SLAM, 09/19/1982, 1, ..., 72;
2  LIN, 9, 2, 500;
3  NETWORK;
4  RESOURCE/ALFA( 1), 1;
5  RESOURCE/BLK1( 1), 2;
6  RESOURCE/BLK2( 1), 3;
7  RESOURCE/BLTA( 1), 4;
8  RESOURCE/BLTA( 5), 5;
9  RESOURCE/PI ( 1), 6;
10  RESOURCE/PHO ( 1), 7;
11  RESOURCE/ZETA( 1), 8;
12  RESOURCE/GAM( 2), 9;
13  CREATE, EXPO( 2.00000), 0, 1, 50;
14  ASSIGN, ATRIB(2) = 1;
15  ACT, , BETA;
16  CREATE, MPSSN( 3.20000), 20, 1, 100;
17  ASSIGN, ATRIB(2) = 2;
18  ACT, , BETA;
19  CREATE, BETA ( 2.00000, 1.00000), 10, 1, 75;
20  ASSIGN, ATRIB(2) = 3;
21  ACT, , PHI;
22  CREATE, GAM( 6.20000, 5.10000), 0, 1, 100;
23  ASSIGN, ATRIB(2) = 4;
24  ACT, , PHI;
25  CREATE, RLOCN( 9.10000, 1.30000), 150, 1, 250;
26  ASSIGN, ATRIB(2) = 5;
27  ACT, , GAM;
28  CREATE, RNORM( 4.10000, 2.60000), 500, 1, 63;
29  ASSIGN, ATRIB(2) = 6;
30  ACT, , ZETA;
31  CREATE, UNFRM( .75000, 1.50000), 0, 1, 200;
32  ASSIGN, ATRIB(2) = 7;
33  ACT, , GAM;
34  CREATE, TRIAG( 2.00000, 5.00000, 10.00000), 15, 1,
35  ASSIGN, ATRIB(2) = 8;
36  ACT, , BETA;
37  CREATE, 1.50000, 1, 1, 85;
38  ASSIGN, ATRIB(2) = 9;
39  ACT, , GAM;
40  ALFA AWAIT( 1/ 50), ALFA;
41  GOON, 1;
42  ACT, , ATRIB(2) .EQ. 4, L101;
43  L101 GOON;
44  ACT, , 3.10000;
45  GOON;
46  ACT, , RLOCN( 2.70000, .33400);
47  REF, ALFA;
48  ACT, , TRM3;
49  BLK1 AWAIT( 2/9999), BLK1;
50  GOON, 1;
51  ACT, , ATRIB(2) .EQ. 1, L102;
52  ACT, , ATRIB(2) .EQ. 2, L103;
53  ACT, , ATRIB(2) .EQ. 3, L104;
54 ACT,,ATRIB(2).EQ.4,L105;
55 ACT,,ATRIB(2).EQ.5,L106;
56 ACT,,ATRIB(2).EQ.7,L107;
57 ACT,,ATRIB(2).EQ.9,L108;
58 L102 COON;
59 ACT,,.50000;
60 FREE,RLK1;
61 L103 COON;
62 FREE,RLK1;
63 ACT,,DLTA;
64 L104 COON;
65 ACT,,.45000,.21000);
66 FREE,RLK1;
67 ACT,,.99000,.21000);
68 FREE,RLK1;
69 ACT,,BETA;
70 L105 COON;
71 ACT,,.99000,.21000);
72 FREE,RLK1;
73 ACT,,BETA;
74 L106 COON;
75 ACT,,.00090;
76 FREE,RLK1;
77 ACT,,.00099;
78 FREE,RLK1;
79 ACT,,BETA;
80 L107 COON;
81 FREE,RLK1;
82 L108 COON;
83 ACT,,.70000;
84 FREE,RLK1;
85 ACT,,.70000;
86 FREE,RLK1;
87 ACT,,.70000;
88 BLK2 AWAIT(3/9999),BLK2;
89 COON,);;
90 ACT,,ATRIB(2).EQ.2,L109;
91 ACT,,ATRIB(2).EQ.4,L110;
92 ACT,,ATRIB(2).EQ.5,L111;
93 ACT,,ATRIB(2).EQ.7,L112;
94 ACT,,ATRIB(2).EQ.9,L113;
95 L109 COON;
96 ACT,,BETA (1.23400,2.34500);;
97 FREE,RLK2;
98 ACT,,BETA;
99 FREE,RLK2;
100 ACT,,BETA;
101 FREE,RLK2;
102 FREE,RLK2;
103 ACT,,BETA;
104 FREE,RLK2;
105 FREE,RLK2;
ACT, TRM1;

GOON;

ACT, PLOGC( 1.33300, .88800);

FREE, PI ;

ACT, RHO;

GOON;

ACT, 6.55000;

FREE, PI ;

ACT, TRM1;

GOON;

ACT, 9.75000;

FREE, PI ;

ACT, GAMMA;

GOON;

ACT, .00010;

GOON;

ACT, TRIAC( .85000, .88000, .91000);

FREE, PI ;

ACT, TRM1;

GOON;

ACT, EXPOS( .10000);

FREE, PI ;

ACT, ZETA;

GOON;

ACT, BETA ( .50000, .10000);

FREE, PI ;

FREE, PI ;

ACT, ZETA;

RHO AWAIt( 10/7), RHO, BLOCK;

GOON, 1;

ACT, ATRIB(2).EQ. 1, L137;

ACT, ATRIB(2).EQ. 2, L138;

ACT, ATRIB(2).EQ. 3, L139;

ACT, ATRIB(2).EQ. 4, L140;

ACT, ATRIB(2).EQ. 5, L141;

ACT, ATRIB(2).EQ. 6, L142;

ACT, ATRIB(2).EQ. 7, L143;

GOON;

ACT, PNSSN( 1.60000);

FREE, RHO ;

FREE, RHO ;

FREE, RHO ;

ACT, ZETA;

GOON;

ACT, UNFRM( 2.10000, 3.60000);

GOON;
FREE,RHO;

ACT,,PI;

1.140 GOON;

ACT,,NPSSN(.70000);

GOON;

ACT,,EXPON(.70000);

FREE,RHO;

ACT,,ZETA;

1.141 GOON;

ACT,,BLOGN(.99900,.88800);

FREE,RHO;

ACT,,ZETA;

1.142 GOON;

ACT,,GAMA(.45000,.23000);

FREE,RHO;

ACT,,ZETA;

1.143 GOON;

FREE,RHO;

ACT,,GAMA;

ZETA,WAIT(8/30),ZETA;

GOON,1;

ACT,,ATRIB(2).EQ.2,1.144;

ACT,,ATRIB(2).EQ.3,1.145;

ACT,,ATRIB(2).EQ.4,1.146;

ACT,,ATRIB(2).EQ.5,1.147;

ACT,,ATRIB(2).EQ.6,1.148;

ACT,,ATRIB(2).EQ.7,1.149;

1.144 GOON;

ACT,,ENORM(2.10000,.44000);

GOON;

FREE,ZETA;

1.145 GOON;

ACT,,ZETA(6.10000,3.70000);

FREE,ZETA;

ACT,,TR42;

1.146 GOON;

FREE,ZETA;

ACT,,TR43;

1.147 GOON;

ACT,,PHIEM(.54000,.62000);

GOON;

ACT,,TRIAC(.00030,2.50000,15.00000);

FREE,ZETA;

ACT,,TR11;

1.148 GOON;

FREE,ZETA;

ACT,,R1TA;

1.149 GOON;

ACT,,25.00000;

GOON;

ACT,,5.00000;

FREE,ZETA;

ACT,,TR12;
GAMA AVAIL( 9/15),GAMA,RALK(BLK2);
GOOK, 1;
ACT,,ATRIB(2).EQ. 2,L150;
ACT,,ATRIB(2).EQ. 4,L151;
ACT,,ATRIB(2).EQ. 5,L152;
ACT,,ATRIB(2).EQ. 7,L153;
ACT,,ATRIB(2).EQ. 9,L154;
L150 GOON;
ACT,,.000000;
GOON;
ACT,,1.10000;
FREE,GAMA;
ACT,,TRI1;
L151 GOON;
FREE,GAMA;
ACT,,ALFA;
L152 GOON;
ACT,,.00010;
GOON;
ACT,,TRIAC(.85000, .88000, .90000);
FREE,GAMA;
ACT,,PI;
L153 GOON;
FREE,GAMA;
ACT,,ZETA;
L154 GOON;
ACT, UNFRM( 4.20000, 6.50000);
GOON;
ACT,,1.621000;
FREE,GAMA;
ACT,,PI;
TRM1 COLCT, INT(1), EXIT INTRVL TRM1;
TERM;
TRM2 COLCT, INT(1), EXIT INTRVL TRM2;
TERM;
TRM3 COLCT, INT(1), EXIT INTRVL TRM3;
TERM;
END;
INIT, 0;
FIN;
SLAM ECHO REPORT

SIMULATION PROJECT SLAM

DATE  9/19/1982

SLAM VERSION JAN 79

BY SLAMPP

RUN NUMBER 1 OF 1

GENERAL OPTIONS

PRINT INPUT STATEMENTS (ILIST): YES
PRINT ECHO REPORT (IECHO): YES
EXECUTE SIMULATIONS (IXQT): YES
PRINT INTERMEDIATE RESULTS HEADING (IPIRH): YES
PRINT SUMMARY REPORT (ISHRY): YES

LIMITS ON FILES

MAXIMUM NUMBER OF USER FILES (MFILS): 9
MAXIMUM NUMBER OF USER ATTRIBUTES (MATR): 2
MAXIMUM NUMBER OF CONCURRENT ENTRIES (MNTRY): 500

FILE SUMMARY

<table>
<thead>
<tr>
<th>FILE NUMBER</th>
<th>INITIAL ENTRIES</th>
<th>RANKING CRITERION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>FIFO</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>FIFO</td>
</tr>
</tbody>
</table>

STATISTICS BASED ON OBSERVATIONS

<table>
<thead>
<tr>
<th>COLCT NUMBER</th>
<th>COLLECTION MODE</th>
<th>IDENTIFIER</th>
<th>HISTOGRAM SPECIFICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NETWORK</td>
<td>EXIT INTRVL TRM1</td>
<td>NCEL HLOW HWID</td>
</tr>
<tr>
<td>2</td>
<td>NETWORK</td>
<td>EXIT INTRVL TRM2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NETWORK</td>
<td>EXIT INTRVL TRM3</td>
<td></td>
</tr>
</tbody>
</table>
RANDOM NUMBER STREAMS

<table>
<thead>
<tr>
<th>STREAM NUMBER</th>
<th>SEED VALUE</th>
<th>REINITIALIZATION OF STREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>124397022910957</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>346713363389</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>7965466614341</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td>18417032136813</td>
<td>NO</td>
</tr>
<tr>
<td>5</td>
<td>280033029935085</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>147959512963949</td>
<td>NO</td>
</tr>
<tr>
<td>7</td>
<td>125894583854829</td>
<td>NO</td>
</tr>
<tr>
<td>8</td>
<td>15047775663725</td>
<td>NO</td>
</tr>
<tr>
<td>9</td>
<td>22787476727917</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>821740779446221</td>
<td>NO</td>
</tr>
</tbody>
</table>

INITIALIZATION OPTIONS

BEGINNING TIME OF SIMULATION (TTREG): 0.
ENDING TIME OF SIMULATION (TTFIN): .1000E+21
STATISTICAL ARRAYS CLEARED (JJCLR): YES
VARIABLES INITIALIZED (JJVAR): YES
FILES INITIALIZED (JJFIL): YES

NSET/QSET STORAGE ALLOCATION

| DIMENSION OF NSET/QSET (NNSET): | 7000 |
| WORDS ALLOCATED TO FILING SYSTEM: | 3000 |
| WORDS ALLOCATED TO INDEXED LIST TAGS: | 0 |
| WORDS ALLOCATED TO NETWORK: | 3042 |
| WORDS AVAILABLE FOR PLOTS/TABLES: | 958 |

INPUT ERRORS DETECTED: 0

EXECUTION WILL BE ATTEMPTED
# SLAM SUMMARY REPORT

SIMULATION PROJECT SLAM

BY SLAMPP

DATE 9/19/1982

RUN NUMBER 1 OF 1

CURRENT TIME .3235E+04

STATISTICAL ARRAYS CLEARED AT TIME 0.

**STATISTICS FOR VARIABLES BASED ON OBSERVATION**

<table>
<thead>
<tr>
<th>EXIT INTERVAL</th>
<th>MEAN VALUE</th>
<th>STANDARD DEVIATION</th>
<th>COEFF. OF VARIATION</th>
<th>MINIMUM VALUE</th>
<th>MAXIMUM VALUE</th>
<th>NO. OF OBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR#1</td>
<td>.563E+02</td>
<td>.158E+03</td>
<td>.280E+01</td>
<td>.172E+01</td>
<td>.113E+04</td>
<td>295</td>
</tr>
<tr>
<td>TR#2</td>
<td>.391E+03</td>
<td>.356E+03</td>
<td>.934E+00</td>
<td>.903E+01</td>
<td>.115E+04</td>
<td>74</td>
</tr>
<tr>
<td>TR#3</td>
<td>.119E+03</td>
<td>.131E+03</td>
<td>.110E+01</td>
<td>.938E+01</td>
<td>.764E+03</td>
<td>238</td>
</tr>
</tbody>
</table>

**FILE STATISTICS**

<table>
<thead>
<tr>
<th>FILE NUMBER</th>
<th>NODE TYPE</th>
<th>AVERAGE LENGTH</th>
<th>STANDARD DEVIATION</th>
<th>MAXIMUM LENGTH</th>
<th>CURRENT LENGTH</th>
<th>AVERAGE WAIT TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AWAIT</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>AWAIT</td>
<td>0.001</td>
<td>0.045</td>
<td>3</td>
<td>0</td>
<td>0.113</td>
</tr>
<tr>
<td>3</td>
<td>AWAIT</td>
<td>2.991</td>
<td>10.750</td>
<td>61</td>
<td>0</td>
<td>92.125</td>
</tr>
<tr>
<td>4</td>
<td>AWAIT</td>
<td>9.904</td>
<td>17.473</td>
<td>50</td>
<td>0</td>
<td>152.550</td>
</tr>
<tr>
<td>5</td>
<td>AWAIT</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>6</td>
<td>AWAIT</td>
<td>4.399</td>
<td>12.353</td>
<td>50</td>
<td>0</td>
<td>28.399</td>
</tr>
<tr>
<td>7</td>
<td>AWAIT</td>
<td>1.181</td>
<td>1.244</td>
<td>10</td>
<td>0</td>
<td>6.967</td>
</tr>
<tr>
<td>8</td>
<td>AWAIT</td>
<td>9.869</td>
<td>13.005</td>
<td>30</td>
<td>0</td>
<td>570.047</td>
</tr>
<tr>
<td>9</td>
<td>AWAIT</td>
<td>0.628</td>
<td>2.909</td>
<td>15</td>
<td>0</td>
<td>3.790</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>4.118</td>
<td>3.395</td>
<td>19</td>
<td>0</td>
<td>1.162</td>
</tr>
</tbody>
</table>

**RESOURCE STATISTICS**

<table>
<thead>
<tr>
<th>RESOURCE NUMBER</th>
<th>RESOURCE LABEL</th>
<th>CURRENT CAPACITY</th>
<th>AVERAGE UTIL</th>
<th>STANDARD DEVIATION</th>
<th>MAXIMUM UTIL</th>
<th>CURRENT UTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALFA</td>
<td>1</td>
<td>.18</td>
<td>.381</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>BLK1</td>
<td>1</td>
<td>.00</td>
<td>.052</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>BLK2</td>
<td>1</td>
<td>.10</td>
<td>.298</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>BETA</td>
<td>1</td>
<td>.29</td>
<td>.451</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>DLTA</td>
<td>5</td>
<td>.05</td>
<td>.294</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>RESOURCE NUMBER</td>
<td>RESOURCE LABEL</td>
<td>CURRENT AVAILABLE</td>
<td>AVERAGE AVAILABLE</td>
<td>MINIMUM AVAILABLE</td>
<td>MAXIMUM AVAILABLE</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PI</td>
<td>1</td>
<td>.46</td>
<td>.498</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>RHO</td>
<td>1</td>
<td>.03</td>
<td>.179</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>ZETA</td>
<td>1</td>
<td>.39</td>
<td>.489</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>GAMMA</td>
<td>2</td>
<td>.17</td>
<td>.481</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESOURCE NUMBER</th>
<th>RESOURCE LABEL</th>
<th>CURRENT AVAILABLE</th>
<th>AVERAGE AVAILABLE</th>
<th>MINIMUM AVAILABLE</th>
<th>MAXIMUM AVAILABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALPHA</td>
<td>1</td>
<td>.8241</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>BLK1</td>
<td>1</td>
<td>.9973</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>BLK2</td>
<td>1</td>
<td>.9016</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>BETA</td>
<td>1</td>
<td>.7149</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>DLTA</td>
<td>5</td>
<td>4.9460</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>PI</td>
<td>1</td>
<td>.5389</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>RHO</td>
<td>1</td>
<td>.9668</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>ZETA</td>
<td>1</td>
<td>.6066</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>GAMMA</td>
<td>2</td>
<td>1.8349</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
VITA

Donald Paul McCanless was born on 31 March 1954 in San Antonio, Texas. He graduated from Highlands High School in 1972 and attended Trinity University where he received a Bachelor of Science degree in computer science in 1976. Following his commission in the United States Air Force from Officers Training School in August, 1977, he served as a programmer/analyst at Headquarters, Strategic Air Command at Offutt Air Force Base, Nebraska. In May, 1981, he entered the School of Engineering, Air Force Institute of Technology at Wright Patterson Air Force Base, Ohio.

Permanent address: 453 Henlo Blvd.
San Antonio, Texas 78223
TACTICAL AIR CONTROL SYSTEM SIMULATION PROGRAM

AUTHOR:
Ronald F. McComb, Captain, USAF

PERFORMING ORGANIZATION NAME AND ADDRESS:
AIR FORCE INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
WRIGHT PATTERSON AFB, OH 45433

CONTROLLING OFFICE NAME AND ADDRESS:

REPORT DATE:
December 1982

NUMBER OF PAGES:
97

MONITORING AGENCY NAME & ADDRESS (IF different from Controlling Office):

SECURITY CLASS. (OF THIS REPORT):
Unclassified

DISTRIBUTION STATEMENT (OF THIS REPORT):
Approved for public release; distribution unlimited.

DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT):

SUPPLEMENTARY NOTES:

KEY WORDS:
Computerized Simulation
SLAM
Communications Networks

ABSTRACT:
The Tactical Air Force Interoperability Group wants to be able to simulate communication networks. The simulation language for alternative modeling (SLAM) was proposed as the basis for the simulation. However, this language requires too much effort to make any changes to the network to be simulated. A Pre-Processor was written in FORTRAN 77 which allows the user to describe the network in a convenient manner, and produces the necessary SLAM source code to perform the simulation.
END
DATE
FILMED
3-83
DTI