

AD-A241 459



NCS TIB 90-14

2



## NATIONAL COMMUNICATIONS SYSTEM

EMP Mitigation also

See page 1- 1 below



EMP Mitigation below & Programers Guide

TECHNICAL INFORMATION BULLETIN 90-14

# QTCM SOFTWARE DOCUMENTATION



## VOLUME I: PROGRAMMER'S MANUAL

NOVEMBER 1990

91-12718



Office of the Manager  
National Communications System  
Washington, D.C. 20305



Nov 1990

Final

QTCM Software Documentation, Volume I:  
Programmer's Manual

C-DCA100-87-C-0063

Booz, Allen & Hamilton, Inc.  
4330 East-West Highway  
Bethesda, MD 20014

National Communications System  
Office of Technology & Standards  
701 S. Court House Road  
Arlington, VA 22204-2199

NCS TIB 90-14

Approved for Public Release; distribution is unlimited.

The EMP Mitigation Program focuses its efforts on the resources of the nation's Public Switched Networks (PSN) because the PSNs comprise the largest, most diverse set of telecommunication assets in the U.S. and are the focus of National Communications System (NCS) National Security & Emergency Preparedness (NS/EP) telecommunications enhancement activities. This volume includes acronyms and references.

221

Electromagnetic Pulse (EMP)  
Public Switched Networks (PSN)  
National Security & Emergency Preparedness (NS/EP)

Unclassified

Unclassified

Unclassified

Unlimited



**NCS TIB 90-14**



# **NATIONAL COMMUNICATIONS SYSTEM**

**TECHNICAL INFORMATION BULLETIN 90-14**

## **QTCM SOFTWARE DOCUMENTATION**

### **VOLUME I: PROGRAMMER'S MANUAL**

**NOVEMBER 1990**

**Office of the Manager  
National Communications System  
Washington, D.C. 20305**

A-1

NCS TECHNICAL INFORMATION BULLETIN 90-14

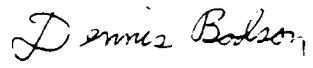
QTCM SOFTWARE MANUAL

OCTOBER 1990

PROJECT OFFICER

  
ANDRE RAUSCH  
Electronics Engineer  
Office of Technology  
and Standards

APPROVED FOR PUBLICATION

  
DENNIS BODSON  
Assistant Manager  
Office of Technology  
and Standards

FOREWORD

The National Communications System (NCS) is an organization of the Federal Government whose membership is comprised of 23 Government entities. Its mission is to assist the President, National Security Council, Office of Science and Technology Policy, and Office of Management and Budget in:

- o The exercise of their wartime and non-wartime emergency functions and their planning and oversight responsibilities.
- o The coordination of the planning for and provision of National Security/Emergency Preparedness communications for the Federal Government under all circumstances including crisis or emergency.

In support of this mission the NCS has initiated and manages the Electromagnetic Pulse (EMP Mitigation Program). The major objective of this program is to significantly reduce the vulnerability of the U.S. telecommunications infrastructure to disabling damage due to nuclear weapon effects in direct support of the survivability and endurability objectives addressed by Executive Order 12472 and National Security Decision Directive 97. Nuclear weapon effects include EMP, magnetohydrodynamic EMP (MHD-EMP), and fallout radiation from atmospheric detonations. The purpose of this Technical Information Bulletin is to provide the reader with information relating to specific areas of EMP which are being investigated in support of the NCS EMP Mitigation Program.

Comments on this TIB are welcome and should be addressed to:

Office of the Manager  
National Communications System  
ATTN: NCS-TS  
Washington, DC 20305-2010  
(202) 692-2124

## TABLE OF CONTENTS

### **VOLUME I - PROGRAMMER'S MANUAL**

1.0 INTRODUCTION	1-1
→ 1.1 EMP Mitigation Program Overview	1-2
1.2 Purpose	1-3
1.3 Organization	1-5
2.0 BACKGROUND	2-1
2.1 History of the QTCM	2-1
2.2 Program Description and Assumptions	2-2
3.0 QTCM ALGORITHM DESCRIPTION	3-1
3.1 Input Formatter	3-1
3.2 Network Simulation	3-5
3.3 Output Formatter	3-8
4.0 MODULE DESCRIPTIONS	4-1
4.1 Input Modules	4-1
4.2 Network Simulation Modules	4-75
4.3 Output Modules	4-131
APPENDIX A	
APPENDIX B	
LIST OF ACRONYMS	
LIST OF REFERENCES	

## LIST OF EXHIBITS

EXHIBIT  
NUMBER

PAGE  
NUMBER

3-1 QTCM Flow Diagram

3-6

## 1.0 INTRODUCTION

National Security Decision Directive (NSDD) 97 and Executive Order (E.O.) 12472 call for the ability to maintain communication capabilities in times of national disaster, which could include a nuclear attack. One of the byproducts of a nuclear detonation is electromagnetic pulse (EMP), which is characterized by intense, high-frequency electromagnetic fields. The currents induced on telecommunications equipment by EMP may be sufficiently severe to damage the telecommunications resources used by critical government users. In an effort to understand and mitigate the detrimental effects of EMP, the Office of the Manager, National Communications System (OMNCS) sponsors the EMP Mitigation Program to analyze, and, where possible, lessen the effects of EMP on national telecommunication resources.

The EMP Mitigation Program focuses its efforts on the resources of the nation's Public Switched Networks (PSNs), because the PSNs comprise the largest, most diverse set of telecommunication assets in the United States and are the focus of National Communications System (NCS) National Security and Emergency Preparedness (NS/EP) telecommunications enhancement activities. Additionally, the majority of NCS member organizations rely on the PSNs for conducting their NS/EP responsibilities. Telecommunications equipment are most susceptible to EMP which occurs when nuclear weapons are detonated higher than 50 kilometers above the earth's surface. This type of EMP is called high-altitude EMP, or HEMP. Throughout the rest of this paper, the term EMP will actually refer to HEMP. In addition to studying the effects of EMP on network performance, the EMP Mitigation Program has been expanded to address the effects of fallout radiation and traffic congestion on the PSNs.



## 1.1 PROGRAM OVERVIEW

The PSNs are vast, complex, commercial resources that encompass the entire nation with sophisticated equipment, and continually change as new technologies are implemented. Since the PSNs are comprised of private companies, the OMNCS has little direct control over how they operate and the equipment they employ. These issues are dictated by the commercial marketplace. Due to proprietary concerns, it is not possible for the government to have complete information regarding the structure and operation of the commercial carrier networks. While the lack of complete data makes it difficult to assess the effects of EMP on the PSNs, it is the belief of the OMNCS that useful preliminary EMP effects estimations on the PSNs can be conducted using the data available. These estimates are useful for developing long-range EMP initiatives.

The EMP Mitigation Program uses an evolutionary process to continually improve its understanding of the EMP effects on the PSNs. The basis for the analyses is the test data regarding the survivability of individual switches and transmission facilities. The OMNCS maintains an ongoing EMP testing program to evaluate the survivability of the major equipment types employed in the PSNs. Presently, the AT&T 4ESS switch is being EMP tested. This equipment survivability information, combined with a knowledge of PSN topologies and the EMP threat, is used to estimate post-attack connectivity at the network level. The key phrase is "network level," because the NCS is concerned with national communication capabilities of the overall network, rather than understanding only the effect on individual network equipment.

In support of these network level analyses, the OMNCS has developed a set of mathematical and computer modeling tools. These tools all interact with one another for a common end result -- to predict the survivability of the PSNs to EMP exposure.

Such network level analysis results can then be used to identify and recommend appropriate EMP mitigation strategies for the PSNs.

In addition to assessing network level EMP effects, the OMNCS analyzes the effects of fallout radiation on the PSNs. Like EMP, fallout radiation is a byproduct of a nuclear detonation, and is also a potential threat to the survivability of network resources. Fallout radiation is the residual radiation that remains in the atmosphere after a nuclear blast, and which can be carried by wind and rain to locations far from the weapon detonation point. A method for assessing the survivability of equipment subjected to fallout radiation based on the survivability of the equipment's piece parts has been developed.

The most recent addition to the EMP Mitigation Program has been to analyze traffic congestion effects on networks degraded by nuclear effects. In such an emergency situation, there is a likelihood that surviving civilian and NS/IP end-users will offer more traffic to the network than it was engineered to handle. This occurrence, in conjunction with damage to network equipment and services by nuclear effects, may result in high call blocking levels for users. This effort and report are devoted to documenting and enhancing the modeling tool used to assess network congestion effects in a damaged network - the Queuing Traffic Congestion Model (QTCM).

## 1.2 PURPOSE

The OMNCS is committed to understanding the effects high traffic loads have on the PSNs in NS/EP conditions. Most previous network level analyses focused on the physical and logical connectivity of PSN switches. However, if the network is

congested, each end-user has to compete with other network users for a limited amount of surviving network resources.

Several recent analyses (References 1, 2, and 3) have been conducted to assess traffic congestion effects. The initial traffic study was performed on the Spanish Post, Telegraph, and Telephone (PTT) network, as the small size of the Spanish network was appropriate for developing traffic modeling capabilities. Two types of network modeling tools were used and compared:

- . Discrete Event - COMNET II.5
- . Queuing - Queuing Traffic Congestion Model (QTCM - formerly IVSN)

QTCM proved to be more effective than COMNET II.5 for evaluating large networks, such as the PSNs. However, the QTCM required many changes and improvements to reflect the modeling requirements of the OMNCS.

Many of these changes were made during an analysis of the US Sprint network. The QTCM was expanded to accommodate 100 switches, eight alternate routes, and a trunk reservation style priority treatment for two user classes. After these changes were validated, the US Sprint network was analyzed.

Further changes were made in conjunction with the most recent traffic study conducted on the AT&T network, the largest PSN. The most significant upgrades made during this analysis were the abilities to incorporate actual EMP damage information from the Network Connectivity Assessment Model (NCAM) and to calculate call waiting times for users in two priority classes. In order to model the AT&T network, the maximum number of switches was raised to 200 switches.

### 1.3 ORGANIZATION

For ease of use, this report is divided into two volumes - a Programmer's Manual and a User's Manual. Volume I is the programmer's manual, and includes acronyms and references. Volume I provides the information necessary to run the QTCM, and serves as a user's manual for the program.

Section 2.0 provides an overall description of the QTCM and its historical background. The algorithm used by the QTCM to model network traffic effects is presented in section 3.0. Each of the three sections of the QTCM -- input formatter, network simulation, and output formatter -- is functionally described. Section 4.0 consists of individual module descriptions for all subroutines and functions in the QTCM.

Volume II, the User's Manual, is designed for the QTCM user who needs to know how to operate the model, but does not need to know the details of the algorithmic calculations. Volume II details the system requirements, initialization procedures, and run-time operating instructions for the QTCM. Included are descriptions of the input and output file formats, execution instructions for the input, model, and output sections, and instructions on how to log on to the system and print output files.

## 2.0 BACKGROUND

This section describes the historical background and principal functions of the QTCM. Included is a discussion of the model's origin and foundations, as well as a brief overview of the model's capabilities. A more detailed description of the simulation algorithms is contained in Section 3.2.

### 2.1 HISTORY OF THE QTCM

The QTCM software is based on the Katz model developed by Steven S. Katz at Bell Telephone Laboratories, Inc. Further information about the Katz model can be found in the transcript of Katz's presentation of "Statistical Performance Evaluation of a Switched Communications Network" given at the Fifth International Teletraffic Congress in New York on June 20, 1967. The Katz model provides an iterative method of calculating the performance parameters of an arbitrarily-sized network.

Software based on the Katz model was developed by the Defense Communications Agency (DCA) to assist in the design of the AUTOVON network. When the NATO Integrated Communications System Management Agency (NICSMA) was faced with a similar problem in the design of the Initial Voice Switched Network (IVSN), DCA made available a copy of the Katz-based software, then called "IVSN model." Modifications were made to the software by the NICSMA and the Supreme Headquarters Allied Powers Europe (SHAPE) Technical Center in the mid-1980's to incorporate network analysis functions and to apply more modern software design techniques to the software. The result of those modifications and enhancements is the version of IVSN originally used by the OMNCS in 1988 in the EMP Mitigation Program. Over the past two years, the OMNCS has made enhancements to the model to make it more efficient and user-friendly, as well as adding functions that customize

the model for NS/EP analyses. For instance, the network design portions of the QTCM are not presently used since the networks of primary interest to the OMNCS, namely the PSNs, are already in place. Due to the large number of revisions, and to avoid ambiguity between the original and updated versions of IVSN, the model has been renamed the Queuing Traffic Congestion Model, or the QTCM.

## 2.2 PROGRAM DESCRIPTION AND ASSUMPTIONS

The capabilities of the QTCM include the ability to subject the network under study to various traffic loads, blocked-call handling schemes, EMP damage effects, and priority user treatments. The effect of each of these variables on network performance can be quantified in statistical terms, such as call completion capability, carried traffic load, and utilization of network trunks. The QTCM uses continuous simulation and queuing theory techniques to distribute traffic throughout the network. The QTCM is written in the FORTRAN programming language, and runs on a Digital Equipment Corporation (DEC) 6310 computer using the Virtual Memory System (VMS) operating system.

The queuing approach used by the QTCM to analyze network traffic adheres to a number of underlying principles and assumptions that are well-accepted by the telecommunications industry as appropriate for traffic engineering analyses. The following principles and assumptions provide the basis of how the QTCM operates:

- All traffic entering the network is assumed to have a Poisson distribution.
- The model utilizes the mean and variance of actual offered traffic. This provides more precise results than models which assume purely random traffic flows.

- The model calculates results for the network when it reaches statistical equilibrium.
- Carried traffic is adequately characterized by the mean and variance of the probability distributions of the number of simultaneous calls on the network.
- The network's trunk group occupancy distributions are statistically independent.
- There is sufficient switching and control equipment at each node to prevent any congestion at the switches. Therefore, all call blocking is modeled to occur at the trunks rather than at the switches.
- Acceptable blocking probabilities are specified for each switch-pair, rather than as a network average. This enables certain critical switches to be provided with a superior grade of service (GOS).
- Upper and lower bounds on the size of each trunk group are considered with respect to the acceptable blocking probability. This simplifies the calculation of allowed traffic loads at the peak usage hour.

### 3.0 QTCM ALGORITHM DESCRIPTION

The QTCM consists of three segments of software - the input formatter, the network simulation, and the output formatter. These segments are discussed in the sections below.

#### 3.1 INPUT FORMATTER

The QTCM input formatter (INPFMT) creates a large, flat input file, which consists of network specifications for the QTCM network simulation segment. Upon execution, INPFMT displays a menu to allow the user to create a new data file, or amend an existing one. The output from INPFMT is an ASCII file that describes the details of the network under study. This file is ready for use by the QTCM network simulation segment. INPFMT executes the following functions:

- . Create a new file using input from the terminal or from an existing file
- . Amend an existing file using terminal input or data from an existing file
- . Display any section of data
- . Validate all sections of data both independently and with cross-checking
- . Create an output file containing the validated input data.

The resulting input data file consists of nine data sections, with several sections containing subsections amounting to 20 separate data areas in all. The nine major data sections with the number of subsections in parentheses are as follows:

- . General network details (1)
- . Network cost details and network distances (2)
- . Routing table (1)



- . Link size constraints (1)
- . Link connectivity requirements for satellite, terrestrial, military-owned media, and overall network (4)
- . Link size details for satellite, terrestrial, and military-owned media (3)
- . Offered load (1)
- . Nodal switch information (1)
- . Damage information, on network users, nodes, satellite connections, terrestrial trunks, and military-owned trunks, as well as preferential routing information (6).

The QTCM input formatter uses a top-down methodology with the main program being INPFMT. Although some of the modules in the input and output formatters have the same name, the functions they perform may be different. This main program is linked with separately compiled subroutines that can be grouped into seven distinct functional areas as follows:

Control code	File handling routines
Display code	Menu routines
Amending code	Low-level routines
Validating code	

The first function, control code, is performed by the modules INPFMT, CONTRL, PRCESS, and NODMGE.

The display function for INPFMT is carried out by subroutines DISPLY, DSPSC1, DSPSC2, DSPSC3, DSPSC4, DSPSC5, DSPSC6, DSPSC7, DSPSC8, DSPSC9, and DSPMTX.

The amending function for INPFMT is performed by the subroutines INPNOD, AMENDR, AMNSC1, AMNSC2, AMNSC3, AMNSC4, AMNSC5, AMNSC6, AMNSC7, AMNSC8, AMNSC9, and AMNMTX.

The validation of input data by INPFMT is executed by the subroutines CHECK, CHKSC1, CHKSC2, CHKSC3, CHKSC4, CHKSC5, CHKSC6, CHKSC7, CHKSC8, CHKSC9, XCHECK, and FNLCHK.

There are also some file-handling functions that must be carried out by the input formatter. Subroutines PSNFIL, CLOSE, GETFIL, GETNAM, INDATA, and OUTDAT execute these functions.

The menu routines for INPFMT are REPLY, MENU1, MENU2, MENU3, MENU4, MENU5, MENU6, and MENU7.

The remaining modules of INPFMT execute support functions such as error messages, character manipulation, or data initialization. These modules consist of ERROR, WARN, BUFFER, CHRINT, INPVAL, INIT, BLKDAT, CLR\_SCREEN, and HEADER.

INPFMT is the main driver program for the QTCM input formatter. It first calls subroutine BLKDAT to initialize variables, data arrays, and common data areas with default values. Then the subroutine INIT is called to retrieve user input. INIT calls MENU 1 to display the first menu, which prompts the user to choose to create a new input file, amend an existing one, or exit. If the user chooses to exit, INPFMT ends.

However, if the user chooses to continue, then CONTRL is called to drive the input formatter in either the AMEND or CREATE mode. These differ in control. In AMEND mode, the user has the modify data in any section of the file already opened by INIT's call to GETFIL. In CREATE mode, the user is forced to enter data in each section in sequence. In either case, PRCESS is called to handle a specific section of data, either chosen by the user or next in sequence. After completing all data edits, FNLCHK is called to finish processing and validation.

INPFMT then calls OUTDAT to write to a named data file any validated section or subsection of data that has been updated. GETNAM is used by OUTDAT to retrieve the user-specified file name. The subroutine CLOSE then closes all files used during the current session. INPFMT makes a final call to INIT to amend another data file or quit the program.

The remaining input modules not mentioned above perform support functions for INPFMT. Some examples of these subroutines and their functions as follows:

REPLY handles all menu responses by calling BUFFER to handle the user input.

PRCOPT handles responses to options when displaying data to a screen.

BUFFER is called to handle user input. It removes leading and trailing spaces and open and close brackets. BUFFER then passes the new version of the input and the length of the input string to the calling routine.

CHRINT is a function that converts characters to their equivalent integer values (e.g. '8' to 8).

ERROR is the error handling routine. It outputs error message text, one or more error values, and uses the alternate return specifier (\*) to set up an error route through the calling process.

### 3.2 NETWORK SIMULATION

The network simulation segment of the QTCM has two primary functions, network design and network performance analysis. The two functions operate autonomously and are both based on the Katz model's iterative method of calculation. In its design mode, the QTCM determines a network configuration that ensures a specified maximum blocking probability and switch-pair GOS. The resultant network configuration consists of trunk capacities and switch-pair routes. The maximum blocking probability is an input variable that the network is engineered to support. Low blocking tolerances require more complex routing procedures and larger trunk sizes.

In network analysis mode, the model calculates the call blocking probabilities, mean GOS, carried traffic loads, and a variety of other performance parameters. These calculated parameters are based on a specified network configuration consisting of trunk sizes, switch-pair connectivity and locations, and switch-pair routing tables.

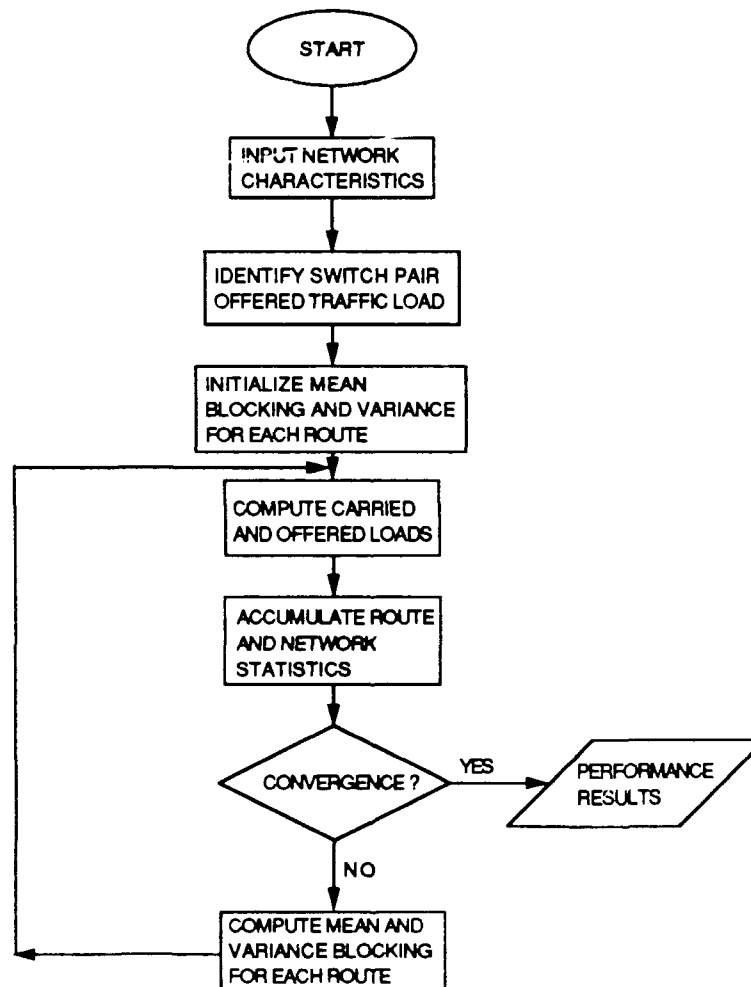
The flow diagram of the analysis portion of the QTCM is presented in Exhibit 3-1. The first step is to input the network characteristics. The network configuration data necessary for the QTCM are the following:

- . Switch locations
- . Switch interconnections
- . Either Erlang B or Erlang C blocked call handling

- . Average offered traffic load between each switch pair
- . Range of acceptable call blocking probabilities for each switch pair
- . Traffic routing (if it is not desired for the model to generate them)
- . Switch pair trunk sizes.

The network traffic load is specified by inputting the offered load, in Erlangs, between each pair of network switches. The network topology information and offered traffic load form the core of the user input data.

EXHIBIT 3-1  
QTCM Flow Diagram



The QTCM is initialized with estimated trunk blocking factors. If the initial blocking values can be chosen to be near the resulting values, the run times for the simulation will be shorter. The model uses routing information, trunk sizing, offered load, and blocking estimates to initiate an origin-to-destination traffic loading process. The QTCM distributes portions of the network traffic, called "parcels," over network trunks. The distribution algorithm initially places traffic parcels over the primary route. Then, overflow traffic is distributed over the number of alternate routes available in the particular simulation.

The size of the parcels, which is in units of Erlangs, is initially large, and is reduced over successive simulation iterations. The parcels are distributed over the trunks and serve as an offered load. Call blocking is calculated on a per trunk basis from the resultant carried load and the network trunk capacity. The model can calculate blocking using either the Erlang B or Erlang C method of handling blocked calls. A series of comprehensive statistics on call blocking is accumulated, including the amount of traffic carried to the destination, use of alternate and primary routes, and the number of links used per connection.

A convergence test is made to see if another iteration is necessary. The convergence process identifies whether the results from the present iteration differ substantially from those of the previous iteration. If the two sets of results are similar, the program has reached a steady state. The program then terminates and outputs the network performance statistics. If convergence has not been attained, another iteration is performed with reduced parcel sizes. These results are again compared with those of the previous iteration. This iterative process continues until convergence is obtained. The number of iterations required is a function of network topology, offered load, alternate routes, estimated initial blocking factors, and required precision.

The QTCM can generate its own routing tables at run time, if desired. It constructs a routing table to support the switch-pair connectivity and routing criteria entered by the user. Routing criteria may be distance, length (in hops), or trunk costs between switches. Based on the chosen routing criterion, a routing algorithm is used to identify the best switch-pair routes. The routing algorithm also attempts to minimize the number of tandem switches that are used in a connection. The current version of the QTCM defines up to eight most effective routes between each connected switch pair.

The network design and routing capabilities of the QTCM have not been used by the OMNCS. However, they illustrate the original purpose and comprehensiveness of the model.

### 3.3 OUTPUT FORMATTER

The QTCM network simulation program produces two output files. One is a formatted log file and the second is the data file, which is unformatted to save space. Therefore, it is necessary to have a program that allows the QTCM user to selectively view the simulation results and to output, or "dump," them to a formatted file for later use.

The output formatter program, OUTFMT, allows the user to execute the following functions:

- . Dump all data to a formatted file (currently not functional)
- . Dump sections of data to a formatted file (currently not functional)
- . Dump subsections of data to a formatted file (currently not functional)
- . Display subsections of data to a terminal

- . Dump a screen of data to a formatted file.

If data is dumped directly to a file, the log file information is also inserted in the file.

The QTCM output formatter has been designed using a top-down methodology with the main program being OUTFMT. It calls OPNFIL to open the required files, copies the log file to the formatted output file and reads the number of nodes and the number of design iterations done. PSNFIL is used to position the file pointer correctly for the above data to be read, and ERROR is called if an error is detected.

A DO-WHILE construct is used in OUTFMT to display the first menu, MENU1, and process the response by either setting the controlling variable of the loop so that the loop is exited or calling DISPLY to control further processing. MENU1 prompts the user to choose to:

- . Dump sections of data to a file
- . Display sections of data
- . Exit.

Once out of the DO-WHILE construct, CLOSE is called to close the output file or delete it if nothing has been written to it.

In DISPLY, if the user has requested to dump data to a file, MENU2 is called to give the following options:

- . Selective dump
- . Dump all the file
- . Exit.



If the first option is chosen, or the user requested to display, MENU3 is called to display the three major sections of data that exist. These sections are as follows:

- . Input data
- . Summary iteration data
- . Detailed analysis data.

MENU4, MENU5, and MENU6 are called for the above options, respectively, to display the subsections within that particular area.

Finally, DSPFIL is called to select the relevant routine to display or dump a subsection of data. For summary iteration data, DSPFIL is called for each iteration. For input data and detailed analysis data, DSPFIL is only called once.

DMPALL is called if the second option is requested. It dumps all information in the file. It loops for each section (three times), sets up the number of iterations that exist for each section (this only applies for the second section), and calls DSPFIL for each iteration.

DSPFIL loops for each subsection requested, calling the relevant routine to dump or display it. The section and subsection numbers are passed to PSNFILE, which finds the required data in the unformatted file. Provided it is found, DSPINP, DSPANL or DSPDSN is called according to the section requested.

The two other menus are MENU7 and MENU8. Two subsections (5 and 6) contain several matrices denoting media. MENU7 allows the user to select a specific area. MENU8 handles the damage control data and the non-preferred routing information. MENU8 allows the user to select a specific area within this subsection.

The remaining modules are the support routines. REPLY handles all responses to menus by calling BUFFER to handle the user input. PROPT handles responses to options when displaying data to a screen. When BUFFER is called to handle user input. It removes leading and trailing spaces and open and close brackets. BUFFER then passes the new version of the input and the length of the input string to the calling routine.

CHRINT is a function that converts characters to their equivalent integer values (e.g. '8' to 8).

ERROR is the error handling routine. It outputs error message text, one or more error values, and uses the alternate return specifier (\*) to set up an error route through the calling process.

Finally, a block data module BLKDAT is used to initialize variables in COMMON areas. These include I/O channel numbers and header information.

## 4.0 MODULE DESCRIPTIONS

Each of the three QTCM segments consists of a main program and associated subroutines and functions. In this report these associated subroutines and functions are generically referred to as modules. In the following sections, the modules for each QTCM segment are described in detail. Information such as inputs, outputs, local variables, and purpose is included for each module.

### 4.1 INPUT MODULES

The input formatter consists of 62 modules, including the main program INPFMT. A data flow diagram of this segment is included in Appendix A, and the detailed module descriptions are given below in alphabetical order.

AMENDR (SECT, SUBSCT) - SUBROUTINE

PURPOSE: CONTROLLING ROUTINE THAT CALLS THE APPROPRIATE ROUTINE TO AMEND THE CURRENT SECTION OF THE INPUT FILE UPON USER REQUEST.

INPUTS: SECT - Section number (Integer).  
SUBSCT - Subsection number (Integer).

OUTPUTS: None.

CALLED BY: PRCESS.

CALLS TO : AMNSC1, AMNSC2, AMNSC3, AMNSC4, AMNSC5, AMNSC6, AMNSC7, AMNSC8, AMNSC9.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

SECT is used as the pointer to the relevant routine and SUBSCT is passed to those routines that require it.

## AMNMTX (MATRIX, TYPE) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO AMEND MATRICES.

INPUTS:     MATRIX - Matrix to be amended (Logical).  
              TYPE - Indicates that all matrix elements are either integer (I) or real (R)  
                  (Character).

OUTPUTS:     MATRIX (Logical).

CALLED BY:   AMNCS2, AMNCS4, AMNSC5, AMNSC6, AMNSC7.

CALLS TO :   MENU6, INPVAL.

### LOCAL VARIABLES:

#### Integer:

IARR(200,200) - Equivalent to the matrix LARR with all integer elements.  
R1, R2 - Row limits.  
C1, C2 - Column limits.  
SD(2) - Source-destination nodes.  
OPT - User's response to menu options.  
IVEC - Parameter from MENU6 which specifies the node of interest.  
IREQ - Number of responses required from the user.  
I, J - Loop counters and pointers.

#### Logical:

END - Set to .TRUE., when user requests to exit.  
LARR(200,200) - Local storage for MATRIX passed from calling routine.

#### Real:

RARR(200,200) - Equivalent to the matrix LARR with all real elements.  
NEWVAL - New value for matrix.

### MODULE LOGIC AND DESCRIPTION:

The first section of code copies the matrix passed as a parameter into the local storage matrix LARR. MENU6 is called with the parameters OPT, IVEC, and \*999 to display options to the user. The user is given the option to exit and, if chosen, OPT is returned from MENU6 with a value of 0. If this occurs, END is set to .TRUE. and the routine AMNMTX is exited.

If OPT is 1, 2, or 3, the user is prompted to enter a default value. INPVAL is called to read the user's response. If the user has not requested to exit and OPT is not equal to 6, the major loop is entered and the row and column limits are set up. The values of the source-destination nodes are displayed, provided a default value is not being used, and the user is prompted to enter the current value of the source-destination pair. Either the value is left alone or a new one is input. The value is then put into the appropriate matrix depending on TYPE. If a default is being used, it is also put into the appropriate array.

Having completed all modifications (from C1 to C2 and R1 to R2), the menu is displayed and the loop repeated if necessary. On a user-requested exit, the local matrix LARR is copied back into MATRIX, the matrix which as passed to AMNMTX.

If OPT equals 6, the user is prompted to enter source-destination nodes or 'E' for exit. INPVAL is called again to read the response.

## AMNSC1 - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO AMEND THE GENERAL NETWORK DATA AND SYSTEM  
              DEFAULT VALUES - SECTION 1.

INPUTS:       None.

OUTPUTS:      Revised general network data and system default values.

CALLED BY:    AMENDR.

CALLS TO :    BUFFER, INPVAL.

### LOCAL VARIABLES:

#### Character:

LINE - Holds user's response to certain queries.  
MEDIA - Defines media type - satellite, terrestrial or military.  
TYPE - Defines routing criteria - cost, length or distance.  
YES - 'YES'

#### Integer:

IVAL1(6) - Equivalent to the first 6 data items of NNOD.  
IVAL2(17) - Equivalent to the other 17 data values of MAXLEN.  
DREQ(2) - Equivalent to the two design requirements - ICON1 and ICON2 (not  
          used).  
SCREEN - Loop control variable.  
LEN - Number of characters in user's response.  
I,J - Loop counters and array pointers.

#### Logical:

ALTER - Statement function - set if the user replies YES to prompts.

#### Real:

RVAL1(6) - Equivalent to the first 6 data items of NNOD.  
RVAL2(17) - Equivalent to the other 17 data values of MAXLEN.

### MODULE LOGIC AND DESCRIPTION:

A loop is entered that prompts the user if alterations are to be done for each of the three screens. If there is a negative response, the prompt for the next screen is sent to the user. When three prompts have been output, the loop is then exited.

The first screen deals with the first eight input variables. These variables are those in COMMON area /SEC1 and the first two variables in COMMON area /DEF1. The second screen deals with the next 8 values from /DEF1, and the third screen with the last 7 in the same common area.

In the majority of cases, the current value of a variable is displayed from within the INPVAL subroutine. INPVAL also accepts the user response. The first parameter indicates data type (1-integer, 2-real); the second indicates type of response (a negative value indicates user can enter <RETURN> for current value to be retained); and the third parameter is the variable in question. If the user's response

matches 'YES', the user is prompted for new values or can skip over them. As with the above variables, the default values are set up in INIT.

The variables that do not fall into this category are the design requirements and the media types. In these cases, BUFFER is used to read in the user's response. The response is held in the variable LINE and compared against values stored in TYPE and MEDIA, respectively. (There doesn't seem to be any error checking here. The user could enter something besides cost, length, or distance.)



AMNSC2 (SUBSCT) - SUBROUTINE

PURPOSE:     ALLOWS USER TO ALTER COST AND DISTANCE MATRICES - SECTION 2.

INPUTS:     SUBSCT - Determines if cost or distance matrix is to be altered, according to the following values (Integer):  
              1 - Cost matrix  
              2 - Distance matrix

OUTPUTS:     Cost or distance matrix to user screen.

CALLED BY:   AMENDR.

CALLS TO :   AMNMTX.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

      This routine calls AMNMTX, passing it either the cost or distance matrix which is determined by SUBSCT. It also passes I, indicating that the elements of the respective matrix are integer.

### AMNSC3 - SUBROUTINE

PURPOSE: CREATES OR AMENDS THE ROUTING TABLE - SECTION 3.

INPUTS: None.

OUTPUTS: Routing table to user screen.

CALLED BY: AMENDR.

CALLS TO : BUFFER, INPVAL.

#### LOCAL VARIABLES:

##### Character:

LINE - Holds user's response to certain questions.

YES - 'YES'

##### Integer:

IARR(8) - Holds information on one source-destination pair.

IANS(2) - Parameter for INPVAL - holds up to two numeric responses.

IREQ - Number of numeric responses required from user.

I - Source node number.

J - Destination node number.

K,L,M - Loop variables and pointers.

LEN - Number of characters in user's response.

##### Logical:

END - Set to .TRUE. when user is finished with this section.

FINISH - Set to .TRUE. when user is finished altering one source-destination pair.

ALTER(N) - Statement function that is set if the user replies 'YES' to prompts.

#### MODULE LOGIC AND DESCRIPTION:

The routing table consists of eight pieces of data. The first four are the alternate paths from the source and the last four are the path lengths of each of the alternates. If an alternate is zero, all subsequent alternates are zero, and the path lengths for all alternates of value zero are 99. Therefore, as soon as a user inputs a zero for an alternate, the rest of the information for that source-destination pair is automatically filled in by the AMNSC3 module.

The module outputs the title first, and then the user is asked whether the routing table is to be created by the network program. BUFFER is called to accept the response. If the response equals 'YES', the logical flag END is set to .TRUE. and the module is exited.

The user is then requested to input a source-destination pair. INPVAL is called to accept the user's response. INPVAL is passed three parameters - the first is set to 1 to indicate an integer response, the second is set to 2 to indicate the values required, and the third parameter holds the two switch pair values. If the user wishes to quit, the second parameter is returned with a value of zero and END is set to .TRUE..

The major loop of the module is entered and continued until END has been set to .TRUE.. Within the loop, the routing table flag IX is set to 1 to indicate that a routing table exists. The next section of code copies the relevant source-destination data from the routing table to local storage array IARR for ease of displaying and amending. The values are then displayed to the user's screen.

The inner loop is controlled by the flag FINISH and is terminated when FINISH is set to .TRUE.. It allows the user to set up alternate routes, mark them as 'down-for-maintenance' and enter the path length for the route. The loop is executed a maximum of eight times. If the user sets a route to zero, all subsequent alternate routes are zero, and the path lengths are set to 99.

The array IARR is then copied back into the routing table and displayed to the user's screen. The user is finally asked to alter another source-destination pair or exit.

AMNSC4 - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO AMEND MINIMUM TRUNK SIZE MATRIX.

INPUTS:       None.

OUTPUTS:     Minimum trunk size matrix to user screen.

CALLED BY:    AMENDR.

CALLS TO :    AMNMTX.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

      This routine passes the minimum trunk size matrix GROUP to AMNMTX, the general matrix handling routine. AMNSC4 indicates to AMNMTX that the matrix elements are all integer.

#### AMNSC5 (SUBSCT) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO CREATE OR AMEND THE MANDATORY CONNECTIVITY MATRIX FOR ANY OF THE THREE MEDIA TYPES.

INPUTS:     SUBSCT - Indicates which of the three media is to be handled according to the following values (Integer):

- 1 - Satellite
- 2 - Terrestrial
- 3 - Military

OUTPUTS:     Mandatory connectivity matrix to user screen.

CALLED BY:   AMENDR.

CALLS TO :   AMNMTX.

LOCAL VARIABLES: None.

#### MODULE LOGIC AND DESCRIPTION:

The routine passes the correct matrix, indicated by SUBSCT, to the general matrix handling routine AMNMTX. AMNSC5 also passes 'I' to AMNMTX, indicating that the matrix is integer.

AMNSC6 (SUBSCT) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO CREATE OR AMEND THE TRUNK SIZE MATRIX FOR EACH OF THE THREE MEDIA.

INPUTS:     SUBSCT - Indicates which of the three media is to be handled according to the following values (Integer):  
                    1 - Satellite  
                    2 - Terrestrial  
                    3 - Military

OUTPUTS:     Trunk size matrix to user screen.

CALLED BY:   AMENDR.

CALLS TO :   AMNMTX.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

      This routine calls the trunk size matrix NTRUNK indicated by SUBSCT to the general handling routine AMNMTX. It also passes the parameter 'I' indicating that the elements of NTRUNK are integer.

AMNSC7 - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO CREATE OR AMEND THE OFFERED LOAD MATRIX.

INPUTS:       None.

OUTPUTS:     Offered load matrix to user screen.

CALLED BY:    AMENDR.

CALLS TO :    AMNMTX.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

      This routine passes the offered load matrix OFLOAD to AMNMTX. AMNSC7 also passes the parameter 'R' to AMNMTX, indicating that the elements of OFLOAD are real.

## AMNSC8 - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO CREATE OR AMEND ANY OF THE SWITCH DETAILS  
              USING A MENU TO SELECT A PARTICULAR AREA.

INPUTS:       None

OUTPUTS:      Switch matrix to user screen.

CALLED BY:    AMENDR.

CALLS TO :    MENU7, CLR\_SCREEN, BUFFER, INPVAL.

### LOCAL VARIABLES:

#### Character:

LINE - Holds user's response to queries.  
YES - Holds 'YES'.

#### Integer:

I,K,X,X1,X2 - Loop control and counters.  
TYPE - Holds user's response to menu options.

#### Logical:

END - Set to .TRUE. when user has finished creating or amending switch details.  
DEF - Set to .TRUE. if user wants to use a default value for all nodes.

### MODULE LOGIC AND DESCRIPTION:

The outside loop for this routine is controlled by the flag END. The loop is repeated until the user requests to exit. MENU7 is called to display the switch options. The variable TYPE is used to hold the user's menu choice, and is set to 0 if the user chooses to exit. When TYPE equals 0, the flag END is set to .TRUE. and the loop is exited.

If the user has not requested to terminate the loop and has selected an area to be amended or set up, the current values are output to the screen. The user is asked if a default value is to be used for all nodes. BUFFER is called to accept the response and is compared to YES. If the user's response is equal to YES, the flag DEF is set to .TRUE. and INPVAL is called to accept the default value, which is returned in NEWVAL.

The next section of code loops through each of the nodes. If DEF is set to .TRUE., the switch values are set to the value held in NEWVAL. Otherwise, the user is prompted to enter a value for each node. INPVAL is used to do this and gives the user the opportunity to skip over values as well.

Having set values for one area, the loop is repeated and the menu of switch options is again displayed.



## AMNSC9 (SUBSCT) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO DAMAGE THE NETWORK AND SET UP NON-PREFERRED ROUTES BETWEEN ANY SOURCE-DESTINATION PAIR.

INPUTS:     SUBSCT - Subsection number (Integer).

OUTPUTS:     Damaged network and non-preferred routes.

CALLED BY:   AMENDR.

CALLS TO :   INPNOD, INDATA, INPVAL, ERROR.

### LOCAL VARIABLES:

#### Character:

MESS - Messages to be displayed.

#### Integer:

ANS - User's response for numeric value.

DTRNK - Holds value for number of trunks to be deleted.

I,,K - Loop counters. (I and J also hold source and destination nodes when dealing with trunk damage.)

IREQ - Flag set to 0 if user wants to exit.

LF - Flag number for current section.

NOARR(200,2) - Equivalent to NOUSER matrix.

SD(2) - Source and destination node array.

#### Logical:

ERR - Initially set to .TRUE.. Only set to .FALSE. if user's responses are valid.

### MODULE LOGIC AND DESCRIPTION:

The first section of code initializes the local variables. The variable LF is set to point to the current section of data, and the flag ERR is set to .TRUE..

The second area of code executes based on the user's choice of subsection to be amended. If SUBSCT is 1 or 2, the user has the ability to damage users or switches at nodes. The code prompts user for a node number or exit. The value for the requested node in the matrix NOARR is then set to zero to indicate damage. INPNOD is called to read the user's response and stores it in the local variable ANS. If the user requests to exit, a value of 0 is returned to the variable IREQ. The loop is repeated until all nodes are processed, and the user requests to exit.

If SUBSCT is 3, 4, or 5, the code to damage trunks is entered. The trunk size matrix is read, if it is not already stored. The user is then repeatedly asked for a source-destination pair and the number of trunks to be deleted until the user requests to exit. If the user chooses to exit, control is returned to the calling routine - AMENDR.

The third area of code allows the user to set up non-preferential routes for any source-destination pair. The user is prompted for a source-destination pair or exit. INPNOD is called to read the user's response and the value for the requested node in the matrix NPREF is set to zero to indicate damage.

## BLKDAT - COMMON DATA SECTION

PURPOSE: TO INITIALIZE THE NECESSARY COMMON DATA SECTIONS, WHICH ARE CALLED BLOCKS IN THE QTCM.

INPUTS: None.

OUTPUTS: None.

CALLED BY: N/A. Common data are used throughout input segment.

CALLS TO: None.

LOCAL VARIABLES: None.

### MODULE LOGIC AND DESCRIPTION:

The BLKDAT section defines all of the text message strings for the section header array SCTHDR and the general information text array (TXT(23)). These text strings consist of the following:

SCTHDR(1) = ' 1. GENERAL NETWORK DETAILS'  
SCTHDR(2) = ' 2. NETWORK COSTS & KILOMETRIC DISTANCES'  
SCTHDR(3) = ' 3. ROUTING TABLE'  
SCTHDR(4) = ' 4. MINIMUM LINK SIZE CONSTRAINTS'  
SCTHDR(5) = ' 5. LINK CONNECTIVITY CONSTRAINTS'  
SCTHDR(6) = ' 6. LINK SIZE DETAILS'  
SCTHDR(7) = ' 7. OFFERED LOAD'  
SCTHDR(8) = ' 8. NODAL SWITCH CONSTRAINTS'  
SCTHDR(9) = ' 9. GENERAL NETWORK DETAILS'  
SCTHDR(10) = '10. EVENTUAL SATELLITE ROUTING'

NOTE: SCTHDR(10) is no longer used.

TXT(1) = 'NUMBER OF NODES IN NETWORK'  
TXT(2) = 'DESIGN OBJECTIVE FOR ARITHMETIC MEAN GOS'  
TXT(3) = 'TOLERANCE LIMIT IN MEETING DESIGN OBJECTIVE'  
TXT(4) = 'DISTRIBUTION CONSTRAINT FOR INDIVIDUAL LINK GOS'  
TXT(5) = '1ST CRITERIA FOR ROUTING TABLE - BASED ON MINIMUM'  
TXT(6) = '2ND CRITERIA FOR ROUTING TABLE - BASED ON MINIMUM'  
TXT(7) = 'MAXIMUM PATH LENGTH BETWEEN SOURCE - DESTINATION'  
TXT(8) = 'NUMBER OF ANALYSIS ITERATIONS'  
TXT(9) = 'MINIMUM PARCEL SIZE INITIALIZED AT'  
TXT(10) = 'MINIMUM VALUE FOR SIZE OF PARCEL'  
TXT(11) = 'MINIMUM VALUE FOR NETWORK STABILITY'  
TXT(12) = 'MAXIMUM NUMBER OF SATELLITE HOPS'  
TXT(13) = 'PERCENTAGE OF OFFERED LOAD'  
TXT(14) = 'WEIGHTING FOR SATELLITE COSTS'  
TXT(15) = 'WEIGHTING FOR MILITARY COSTS'  
TXT(16) = 'CHANGES TO NETWORK CONFIGURATION CONSTRUED ON'

TXT(17) = 'LINK BLOCKING FACTOR'  
TXT(18) = 'LINK VARIANCE BLOCKING FACTOR'  
TXT(19) = 'MINIMUM BLOCKING FACTOR'  
TXT(20) = 'TRUNK CREATION PARAMETER'  
TXT(21) = 'E1FRAC - DISTRIBUTION FACTOR'  
TXT(22) = 'E2FRAC - DISTRIBUTION FACTOR'  
TXT(23) = 'MAXIMUM NUMBER OF DESIGN ITERATIONS'

## **BUFFER(TYPE, BUFF, DATALN) - SUBROUTINE**

**PURPOSE:** READS IN ONE LINE OF DATA FROM USER, CHECKS FOR <RETURN>, AND STRIPS OFF LEADING SPACES AND FIRST OPENING BRACKET AS WELL AS ANY CLOSING BRACKETS AT THE END OF THE INPUT.

**INPUTS:** TYPE - Pointer to PROMPT message (Integer).  
BUFF - User input of 1 to 80 characters (Character).  
DATALN - Number of characters in BUFF (Integer).

**OUTPUTS:** None.

**CALLED BY:** FNLCHK, OUTDAT, GETNAM, DSPSC1, DSPSC3, DSPSC8, DSPSC9, DSPSCA, AMNSC1, AMNSC3, AMNSC8, INPNOD, AMNSCA, INPVAL, REPLY, WARN, ERROR, CONTRL.

**CALLS TO :** ERROR.

### **LOCAL VARIABLES:**

#### **Character:**

CH - Used as a pointer to BUFF.  
ICHAR - FORTRAN function to convert character to integer.  
LINE - Input buffer.  
PROMPT(4) - Prompt messages.

#### **Integer:**

BUFLN - Maximum length of BUFF.  
I, J, K - Loop variables and pointers.  
ICH - Numerical value of BUFF character.  
MAXLEN - Maximum length of LINE.

#### **Logical:**

FOUND - Set to .TRUE. when the start of the input string is found.  
END - Set to .TRUE. at the end of processing.

### **MODULE LOGIC AND DESCRIPTION:**

The variables END and DATALN are initialized to .FALSE. and zero, respectively. The input buffer LINE is cleared. If TYPE falls in the range of 0 to 3, a prompt message is output to the user's screen and the response is read into LINE. ERROR is called if <RETURN> is used when input is required (TYPE = 2), otherwise END is set to .TRUE. as no further processing is required.

The code searches the string for the first character that is not a space or an open bracket. This is deemed to be the start of the string and FOUND is set to .TRUE.. If the pointer I reaches MAXLEN, ERROR is called. Once the start of the string is located, J is used to find the end and is deemed to have found the end when the character is neither a space nor a close bracket. DATALN is then calculated using I and J. Provided input exists, it is then copied from LINE into BUFF.

## CHECK (SECT, SUBSCT, OK, \*) - SUBROUTINE

PURPOSE: CONTROLLING MODULE FOR THE VALIDATION ROUTINES FOR EACH SECTION.

INPUTS: SECT - Section number (Integer).  
SUBSCT - Subsection number (Integer).  
OK - Set to .TRUE. if the data in the subsection is valid (Logical).  
\* - Alternate return specifier (Integer).

OUTPUTS: The flag OK set to .TRUE. when the section has been validated.

CALLED BY: PRCESS, FNLCHK.

CALLS TO: CHKSC1, CHKSC2, CHKSC3, CHKSC4, CHKSC5, CHKSC6, CHKSC7, CHKSC8, CHKSC9.

LOCAL VARIABLES: NONE.

### MODULE LOGIC AND DESCRIPTION:

The module calls the relevant validation routine depending on the value of SECT and, where necessary, it passes SUBSCT so that a particular subsection is indicated. The alternate return specifier is also passed to every routine and is used if the data are not valid.

# CHKSC1 (\*) - SUBROUTINE

PURPOSE:     VALIDATES GENERAL INFORMATION - SECTION 1

INPUTS:       \* - Alternate return specifier (Integer).

OUTPUTS:      Error or warning message sent to user if section 1 is invalid.

CALLED BY:    CHECK.

CALLS TO :    ERROR, WARN.

LOCAL VARIABLES:

Real:

DMIN - Minimum value for grades of service.

DMAX - Maximum value for grades of service.

## MODULE LOGIC AND DESCRIPTION:

This subsection verifies that the following variables fall within the given ranges. If the variable is outside the range an error or warning message is sent to the user.

<u>VARIABLE</u>	<u>RANGE</u>	<u>RESULT IF VERIFICATION FAILS</u>
DGOS	>= 0.0	ERROR
DGOS	DMIN to DMAX	WARNING
DSNCNT	1.0 to 9999	ERROR
E1FRAC	0.0 to 1.0	ERROR
E2FRAC	0.0 to 1.0	ERROR
EGOS	>= 0.0	ERROR
EGOS	> 0.0	WARNING
EMIN	0.0 to 5.0	ERROR
LBF	0.0 to 1.0	ERROR
LVBFB	0.0 to 1.0	ERROR
MAXLEN	1.0 to 15.0	ERROR
MAXSAT	0.0 to 2.0	ERROR
NNOD	2.0 to 200.0	ERROR
NOITS	> 4.0	ERROR
PARC	> 0.0	ERROR
RGOS	0.0 to 1.0	ERROR
RGOS	DMIN to DMAX	WARNING
STABLE	0.0 to 1.0	ERROR
TINY	0.0 and 1.0	ERROR
TINYBF	0.0 and 1.0	ERROR

## CHKSC2 (SUBSCT, \*) - SUBROUTINE

PURPOSE: CHECKS COST AND DISTANCE MATRICES FOR SYMMETRY AND POSITIVE VALUES - SECTION 2.

INPUTS: SUBSCT - Subsection number (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: Error message sent to user if section 2 is invalid.

CALLED BY: CHECK.

CALLS TO : ERROR.

### LOCAL VARIABLES:

Integer:

I,J - Loop control variables.  
IFAC - Factorizing value.

Real:

ERR - Error value.

### MODULE LOGIC AND DESCRIPTION:

The first section of code checks to see if SUBSCT equals 1. If this is true, the network cost matrix is validated. Otherwise, the kilometeric distance matrix is validated.

Validation is identical for both the network cost matrix and the kilometeric distance matrix. These two subsections are valid if the following criteria are met:

- (1) All values must be greater than or equal to zero; otherwise, an error message is sent to the user.
- (2) The matrices must be symmetric; otherwise, an error message is sent to the user.

### CHKSC3 (\*) - SUBROUTINE

PURPOSE:     VALIDATES ROUTING TABLE DATA - SECTION 3.

INPUTS:     \* - Alternate return specifier (Integer).

OUTPUTS:     Error message sent to user if section 3 is invalid.

CALLED BY:   CHECK.

CALLS TO :   ERROR.

#### LOCAL VARIABLES:

Integer:

I,J,K,L - Loop and index variables.

R - Holds the four alternate routes for a particular source destination pair.

P - Holds the associated path lengths for R.

IFAC - Factorizing value.

Real:

ERR - Error value.

#### MODULE LOGIC AND DESCRIPTION:

First, the flag IX is checked to determine if a routing table exists. If IX equals 0, then no routing table is present, and control is passed back to the calling routine. If IX does not equal 0, a routing table does exist and CHKSC3 begins a validation of the table. For each source-destination pair, the alternate routes and associated path lengths are read into the two local arrays R and P respectively.

If the source and destination are the same, all routes must be zero and all path lengths must be 99. If this is not the case, an error message is sent to the user. In all cases, if a route is zero, its path length is 99. Routes are zero whenever a source and destination pair are not connected. For all other combination of pairs, the alternate routes cannot point back to the source node or a non-existent node, and the path length cannot exceed the maximum path length (MAXLEN).

As soon as an alternate route is determined to be zero, all subsequent routes are set to be zero and the path lengths are set to be 99.

The final test is that all routes for one pair must be unique. If they are not unique, an error message is sent to the user.



CHKSC4 (\*) - SUBROUTINE

PURPOSE:     VALIDATES THE MINIMUM TRUNK GROUP MATRIX - SECTION 4.

INPUTS:       \* - Alternate return specifier (Integer).

OUTPUTS:     Error message sent to user if section 4 is invalid.

CALLED BY:   CHECK.

CALLS TO :   ERROR.

LOCAL VARIABLES:

Integer:

    I,J - Loop control and array pointers.

    IFAC - Factorizing value.

Real:

    ERR - Error value.

MODULE LOGIC AND DESCRIPTION:

    This routine loops through the matrix, and calls ERROR if any value is less than zero, and then loops through again checking the matrix for symmetry.

# CHKSC5 (S, \*) - SUBROUTINE

PURPOSE: VALIDATES MANDATORY CONNECTIVITY MATRICES - SECTION 5.

INPUTS: S - Subsection number (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: Error message sent to user if section 5 is invalid.

CALLED BY: CHECK.

CALLS TO : ERROR, WARN.

## LOCAL VARIABLES:

Integer:

I,J - Loop control and array pointers.  
IFAC - Factorizing value.

Real:

ERR - Error value.

## MODULE LOGIC AND DESCRIPTION:

The parameter S determines which matrix is to be validated. In the first section of code, the module verifies that the values are 0, 1, or 2. Each of these values has a different meaning as shown below:

- 0 - Do not want the link to exist
- 1 - Link must exist
- 2 - Link existence does not matter.

If the values are other than 0, 1 or 2, an error message is sent to the user.

The second section of code checks that 0 and 1 do not appear for the same point-to-point pair. If they do, WARN is called and the user can accept or reject the conflict.

#### CHKSC6 (SUBSCT, \*) - SUBROUTINE

PURPOSE: VALIDATES ONE OF THE TRUNK SIZE MATRICES DEPENDING ON THE  
PARAMETER SUBSCT - SECTION 6.

INPUTS: SUBSCT - Subsection number (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: Error message sent to user if section 6 is invalid.

CALLED BY: CHECK.

CALLS TO : ERROR.

#### LOCAL VARIABLES:

Integer:

I,J - Loop control and array pointers.  
IFAC - Factorizing value.

Real:

ERR - Error value.

#### MODULE LOGIC AND DESCRIPTION:

The module loops through the trunk size matrix to verify that all values are greater than or equal to zero. If a negative value is found, the ERROR subroutine is called to send an error message to the user's screen. The module then checks that the matrix is symmetrical. If this is not the case, ERROR is called and an error message is displayed.

## CHKSC7 (\*) - SUBROUTINE

PURPOSE:     VALIDATES THE OFFERED LOAD MATRIX - SECTION 7.

INPUTS:       \* - Alternate return specifier (Integer).

OUTPUTS:      Error message sent to user if section 7 is invalid.

CALLED BY:    CHECK.

CALLS TO :    ERROR, WARN.

### LOCAL VARIABLES:

Integer:

I,J - Loop control and matrix pointers.

IFAC - Factorizing value.

Real:

ERR - Error value.

OFLMIN - Minimum value before a warning message is output.

TOTOFF - Total offered load.

### MODULE LOGIC AND DESCRIPTION:

The module first checks that for each node link of the offered load matrix with the value of zero has I and J values that are equal. Otherwise ERROR is called, and an error message is sent to the user's screen. Next, the CHKSC7 module checks that the offered load values are greater than or equal to zero. If not, an error message is sent to the user. CHKSC7 then compares the values to the minimum value OFLMIN. If the value is less than or equal to OFLMIN, then a warning message is sent to the user. Finally, the CHKSC7 subroutine verifies that TOTOFF is not equal to zero.

## CHKSC8 (\*) - SUBROUTINE

PURPOSE: VALIDATES ALL SWITCH DATA - SECTION 8.

INPUTS: \* - Alternate return specifier (Integer).

OUTPUTS: Error message sent to user if section 8 is invalid.

CALLED BY: CHECK.

CALLS TO : ERROR, WARN.

### LOCAL VARIABLES:

Integer:

I - Loop counter and array pointer.

Real:

ERR - Error value.

### MODULE LOGIC AND DESCRIPTION:

The module loops through the switch data and performs the following tests:

<u>TESTED PARAMETER</u>	<u>ACCEPTABLE RANGE</u>	<u>RESULT IF VERIFICATION FAILS</u>
Number of alternates	Integer Between 1 and 8	ERROR
Protocol	Either 1 or 2	ERROR
Maximum number of groups	Greater than 0 otherwise Value other than 1	ERROR WARN
Minimum number of groups	Greater or equal to 0 otherwise Less than or equal to maximum no. of groups otherwise Value other than 0	ERROR ERROR WARN
Maximum number of channels	Greater than or equal to 0 otherwise Value other than 0 otherwise Greater than twice the minimum number of groups	ERROR WARN WARN

# CHKSC9 (SUBSCT, \*) - SUBROUTINE

PURPOSE: VALIDATES DAMAGE AND PREFERENTIAL ROUTING DATA - SECTION 9.

INPUTS: SUBSCT - Subsection number (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: Error message sent to user if section 9 is not valid.

CALLED BY: CHECK.

CALLS TO : ERROR, INDATA.

## LOCAL VARIABLES:

Integer:

I,J,K - Loop counters and pointers.  
SUBSCT - Subsection number.  
LF - Flag number for current section.  
NOARR(25,2) - Equivalent to NOUSER matrix.

Logical:

NOK - Set to .FALSE. when surviving users, switches, or trunks are found.

## MODULE LOGIC AND DESCRIPTION:

If SUBSCT is 1 or 2, the code checks that all users or all switches are not destroyed. NOK is the controlling variable which is set to .FALSE. if a surviving user or switch is found. If no survivors are found, an error message is sent to the user.

For SUBSCT equal to 3, 4, or 5, the code checks that some trunks remain. The first section verifies that all trunk sizes and trunk damage have been read. NOK is again used as the controlling variable. If surviving trunks are found, NOK is set to .FALSE. and control is passed back to the calling routine. Otherwise, an error message is sent to the user indicating that processing cannot continue with all trunks damaged.

# CHRINT(CHVAL) - FUNCTION

PURPOSE: CONVERTS A CHARACTER TO ITS INTEGER EQUIVALENT.

INPUTS: CHVAL - Character to be converted (Character).

OUTPUTS: Integer equivalent.

CALLED BY: INPVAL, PSNFIL.

CALLS TO : None.

## LOCAL VARIABLES:

Character:

TABLE(14) - Look-up table.

Integer:

I - Pointer for table.

## MODULE LOGIC AND DESCRIPTION:

CHRINT is initialized to 99. The table values are compared with the character CHVAL. If found, CHRINT is set equal to the value at that table position.

## CLOSE - SUBROUTINE

PURPOSE: CLOSING ALL FILES OPENED DURING THE CURRENT SESSION.

INPUTS: None.

OUTPUTS: Closed data files.

CALLED BY: INPFMT.

CALLS TO: None.

### LOCAL VARIABLES:

Integer:

I - Loop variable for loop to close all files.

### MODULE LOGIC AND DESCRIPTION:

The module loops through all open files by using the I/O unit numbers held in the array CHAN. Each file is closed until the last one indicated by FPTR is reached. Finally, I/O unit 99 is closed. This I/O unit was used to create and write the output file.



# CLR\_SCREEN - SUBROUTINE

PURPOSE:     CLEARS THE TERMINAL SCREEN.

INPUTS:      None.

OUTPUTS:     None.

CALLED BY:   DSPSC1, DSPSC3, DSPSC8, DSPSC9, DSPMTX, AMNSC8, MENU1, HEADER.

CALLS TO :   None.

LOCAL VARIABLES: None.

## MODULE LOGIC AND DESCRIPTION:

      This routine clears the terminal screen with a format statement that initiates a series of  
<RETURN>.

## CONTRL (QUIT) - SUBROUTINE

PURPOSE: CONTROLS THE AMENDING ROUTINES.

INPUTS: QUIT (Logical).

OUTPUTS: Returns parameter QUIT set to false, if output file is to be created.

CALLED BY: INPFMT.

CALLS TO: MENU3, HEADER, BUFFER, NODMGE, PRCESS, FNLCHK.

### LOCAL VARIABLES:

#### Character:

LINE - The user-entered choice of whether damage is to be assessed.  
NO - Default value for LINE.

#### Integer:

LEN - Length of user-entered choice of whether damage is to be assessed (1 or 2).

#### Logical:

USREND - Inside loop control variable. Set to .TRUE. when sector counter, SECT, exceeds the maximum number of sections, NOSCTS.  
ALLOK - Outside loop control variable. Set to .TRUE. when valid data exists for every section.  
QUIT - When set to .FALSE., an output file is to be created.  
LPRCS - When set to .FALSE., no damage is imposed on the modeling network.

### MODULE LOGIC AND DESCRIPTION:

CONTRL determines which section of the data file currently under review is to be amended. CONTRL then calls PRCESS to handle the editing, and when requested it calls FNLCHK to do final data validation. The subroutine consists of two loops. The inner loop is controlled by USREND. USREND is set to .TRUE. when the pointer to the current section of data, SECT, exceeds the maximum number of sections, NOSCTS. MENU3 is called within this loop to display the sections. It is passed the parameter SECT and the label 999. If the user wants to save the revised data, SECT is returned with a value of 0. If the user "quits," the alternate return route is used and the program jumps to the label 999. If SECT does not exceed the maximum number of sections, PRCESS is called to handle the required section.

Once USREND is set to .TRUE., CONTRL accesses the outer loop, which is controlled by ALLOK. ALLOK is set to .TRUE. when all data is entered and validated for every section. This validation takes place in the subroutine FNLCHK. AMEND is set to .TRUE. in case ALLOK is not set, so that MENU3 is called to display the sections.

QUIT is reset to .FALSE., so that an output file is created.

#### DISPLY (SECT, SUBSCT) - SUBROUTINE

PURPOSE: CONTROLS WHICH DISPLAY ROUTINE IS CALLED FOR EACH SECTION OF DATA.

INPUTS: SECT - Section number (Integer).  
SUBSCT - Subsection number (Integer).

OUTPUTS: None.

CALLED BY: PRCESS.

CALLS TO : DSPSC1, DSPSC2, DSPSC3, DSPSC4, DSPSC5, DSPSC6, DSPSC7, DSPSC8, DSPSC9.

LOCAL VARIABLES: None.

#### MODULE LOGIC AND DESCRIPTION:

DISPLY is the controlling module that calls the relevant display routine for each section of data. SECT is used as the pointer to the relevant display routine and SUBSCT is passed as a parameter to those routines requiring it.

## DSPSCI - SUBROUTINE

PURPOSE:     DISPLAYS GENERAL NETWORK VALUES AND SYSTEM DEFAULT VALUES IN  
              THREE SEPARATE SCREENS.

INPUTS:       None.

OUTPUTS:      General network and system default values to user screen.

CALLED BY:    DISPLY.

CALLS TO :    CLR\_SCREEN, BUFFER.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy parameter for BUFFER.

TITLE1 - Header for all screens.

UNDRLN - Underlines the header.

MEDIA - Defines the three media types - satellite, military, and terrestrial.

TYPE - Holds design criteria.

#### Integer:

CRITER - Equivalent to ICON1, ICON2 (design criteria)

DUMMY2 - Dummy parameter for BUFFER.

### MODULE LOGIC AND DESCRIPTION:

This routine outputs all variables in section 1 on three sequential screens. The two design criteria values (ICON1, ICON2) are used to index the array TYPE to output the media type. Buffer is called at the end of each screen to allow the user to enter a carriage return before proceeding to the next screen. An example of the section 1 display screens is shown below.

### SECTION - 1 GENERAL PROGRAM VARIABLES

NUMBER OF NODES IN NETWORK	: 179
DESIGN OBJECTIVE FOR ARITHMETIC MEAN GOS	: 0.0000
TOLERANCE LIMIT IN MEETING DESIGN OBJECTIVE	: 0.0010
DISTRIBUTION CONSTRAINT FOR INDIVIDUAL LINK GOS	: 0.0010
1ST CRITERIA FOR ROUTING TABLE - BASED ON MINIMUM	: DISTANCE
2ND CRITERIA FOR ROUTING TABLE - BASED ON MINIMUM	: LENGTH
MAXIMUM PATH LENGTH BETWEEN SOURCE - DESTINATION	: 2
NUMBER OF ANALYSIS ITERATIONS	: 100

**SECTION - 1 GENERAL PROGRAM VARIABLES**

MINIMUM PARCEL SIZE INITIALIZED AT	: 0.0100
MINIMUM VALUE FOR SIZE OF PARCEL	: 0.0002
MINIMUM VALUE FOR NETWORK STABILITY	: 0.0001
MAXIMUM NUMBER OF SATELLITE HOPS	: 0
PERCENTAGE OF OFFERRED LOAD	: 100.000 %
WEIGHTING FOR SATELLITE COSTS	: 1.0000
WEIGHTING FOR MILITARY COSTS	: 1.0000
CHANGES TO NETWORK CONFIGURATION CONSTRUED ON	: PTT

**SECTION - 1 GENERAL PROGRAM VARIABLES**

LINK BLOCKING FACTOR	: 0.0100
LINK VARIANCE BLOCKING FACTOR	: 0.0100
MINIMUM BLOCKING FACTOR	: 0.0000
TRUNK CREATION PARAMETER	: 0.7000
E1FRAC - DISTRIBUTION FACTOR	: 0.5000
E2FRAC - DISTRIBUTION FACTOR	: 0.9000
MAXIMUM NUMBER OF DESIGN ITERATIONS	: 1

## DSPSC2 (SUBSCT) - SUBROUTINE

PURPOSE:     DISPLAYS EITHER THE NETWORK COST MATRIX OR THE KILOMETRIC DISTANCE MATRIX.

INPUTS:      SUBSCT - Subsection number (Integer).

OUTPUTS:     Cost and Distance matrices.

CALLED BY:   DISPLY.

CALLS TO :   DSPMTX.

### LOCAL VARIABLES:

Character:

TITLE - Array that holds titles for both cost and distance matrices.

### MODULE LOGIC AND DESCRIPTION:

This module calls DSPMTX passing it either the cost or distance matrix, the matrix title, and a flag indicating that all elements of the matrix are integer. SUBSCT indicates which matrix to pass.

#### SECTION 2 - NETWORK COSTS & DISTANCES SUBSECTION 1 - NETWORK COSTS

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	1	1	1	1	...	1	1	1	1
2:	1	0	1	1	1	...	1	1	1	1
3:	1	1	0	1	1	...	1	1	1	1
27:	1	1	1	1	1	...	1	1	0	1
28:	1	1	1	1	1	...	1	1	1	0

#### SECTION 2 - NETWORK COSTS & DISTANCES SUBSECTION 2 - KILOMETRIC DISTANCES

S/D	1	2	3	4	...	25	26	27	28
1:	0	530	184	328	...	466	640	2039	281
2:	530	0	697	587	...	862	900	2175	541
3:	184	697	0	460	...	323	720	2110	290
27:	2039	2175	2110	1725	...	2420	1400	0	2319
28:	281	541	290	600	...	321	921	2319	0

### DSPSC3 - SUBROUTINE

PURPOSE:     DISPLAYS ROUTING TABLE.

INPUTS:      None.

OUTPUTS:     Routing table to screen.

CALLED BY:   DISPLY.

CALLS TO :   CLR\_SCREEN, BUFFER, INPVAL, ERROR.

#### LOCAL VARIABLES:

Character:

    DUMMY1 - Dummy parameter for BUFFER.

Integer:

    IARR - Matrix which stores information from routing table.

    IANS - Parameter for INPVAL.

    NL - Array which holds the number of lines to be displayed.

    DUMMY2 - Dummy parameter for BUFFER.

    I,J - Loop control variables.

Logical:

    END - Set to .TRUE. when user requests to exit.

#### MODULE LOGIC AND DESCRIPTION:

The first section of code checks that a routing table exists. If no routing table exists the variable IX is set to 0, and a message is sent to the user stating that "There is no routing table." The user then hits <RETURN> to continue. The subroutine BUFFER is called for this purpose. If a routing table does exist, a test is made on the number of nodes. If the number of nodes exceeds 18, the information to be displayed is split over the two screens. The array NL holds the total number of lines to be displayed in both screens.

The user is then prompted to enter the source node of interest or has the option to exit from the display process. INPVAL is the routine that deals with the user's response. If the variable IREQ is equal to 0 after the call, the user has requested to exit and the logical flag END is set to .TRUE..

If the use enters a source node (END is still set to .FALSE.), then the routine then enters a loop that repeatedly displays information until the user requests to exit. ISRCE is set equal to the users response. It is validated and ERROR is called if it is an illegal node number. The user is then prompted to enter a single destination node or 0 for all of them. INPVAL is called again to deal with the user's response. The response is validated and ERROR is called if there is an illegal response. If destination equals 0, then the loop variables, L1 and L2, are set for all destinations. After the source and destination nodes are specified by the user, the routine then enters another loop which stores all the information in IARR from the routing table. The second loop then displays the information in either one or two screens. If a second screen is necessary, the user is prompted to enter <RETURN> before the second screen is displayed.

Finally, the user is prompted to either enter another source node or 'E' to exit. If the user selects 'E', END is set to .TRUE. and the loop is exited. An example of the routing table display screen is shown below.

#### ROUTING TABLE DETAILS

1	1	0	0	0	0	0	0	0	0	99	99	99	99	99	99	99
2	1	1	14	6	22	8	28	7	3	1	2	2	2	2	2	2

.

27	1	1	26	6	4	17	22	5	12	14	2	2	2	2	2	2
28	1	1	7	3	21	6	16	25	22	1	2	2	2	2	2	2

.

. [For each of the 28 nodes]

.

1	2	2	14	6	22	8	28	7	3	1	2	2	2	2	2	2
2	2	0	0	0	0	0	0	0	99	99	99	99	99	99	99	99

.

27	2	2	12	17	14	5	22	26	4	1	2	2	2	2	2	2
28	2	2	7	8	14	21	1	16	22	1	2	2	2	2	2	2



#### DSPSC4 - SUBROUTINE

PURPOSE: DISPLAYS MINIMUM TRUNK SIZE MATRIX.

INPUTS: None.

OUTPUTS: Minimum trunk size matrix to user screen.

CALLED BY: DISPLY.

CALLS TO : DSPMTX.

#### LOCAL VARIABLES:

Character:

TITLE - Holds title to be displayed.

#### MODULE LOGIC AND DESCRIPTION:

This routine passes three parameters to DSPMTX. The first parameter, GROUP, is the minimum trunk size matrix. The second parameter, TITLE, passes the heading "MINIMUM TRUNK SIZE," and the third parameter, I, indicates that the matrix holds elements that are integers.

The diagram below shows an example of a minimum trunk size matrix.

#### SECTION 4 - MINIMUM TRUNK GROUP

S/D	1	2	3	4	5	6	7		25	26	27	28
1:	0	1056	840	1440	312	1224	792	...	792	432	408	432
2:	1056	0	480	1320	312	720	744	...	768	336	384	456
3:	840	480	0	720	168	456	408	...	840	168	144	216
								.				
								.				
27:	408	384	144	672	384	216	192	...	432	216	312	214
28:	432	456	216	552	168	264	1008	...	504	120	214	312

## DSPSC5 (SUBSCT) - SUBROUTINE

PURPOSE:     DISPLAYS THE MANDATORY CONNECTIVITY MATRICES FOR SATELLITE, TERRESTRIAL AND MILITARY-OWNED TERRESTRIAL MEDIA.

INPUTS:     SUBSCT - Indicates which mandatory connectivity matrix is to be displayed, as shown below:

- 1 - Satellite
- 2 - Terrestrial
- 3 - Military

OUTPUTS:     Mandatory connectivity matrices to user screen.

CALLED BY:   DISPLY.

CALLS TO :   DSPMTX.

LOCAL VARIABLES:

Character:

TITLE - Holds titles for all three media.

MODULE LOGIC AND DESCRIPTION:

This routine calls DSPMTX to display the requested matrix using SUBSCT as the pointer to the relevant starting point in MNDCNC, the mandatory connectivity matrix. Shown below is an example display of the mandatory connectivity matrix.

### SECTION 5 - MANDATORY TRUNK CONNECTIONS

### SUBSECTION 1 - SATELLITE

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	0	0	0	0	...	0	0	0	0
2:	0	0	0	0	0	...	0	0	0	0
3:	0	0	0	0	0	...	0	0	0	0
						.				
						.				
						.				
27:	0	0	0	0	0	...	0	0	0	0
28:	0	0	0	0	0	...	0	0	0	0

### SUBSECTION 2 - TERRESTRIAL

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	1	1	1	1	...	1	1	1	1
2:	1	0	1	1	1	...	1	1	1	1
3:	1	1	0	1	1	...	1	1	1	1
						.				
						.				
						.				
27:	1	1	1	1	1	...	1	1	0	1
28:	1	1	1	1	1	...	1	1	1	0

### SUBSECTION 3 - MILITARY

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	0	0	0	0	...	0	0	0	0
2:	0	0	0	0	0	...	0	0	0	0
3:	0	0	0	0	0	...	0	0	0	0
						.				
						.				
						.				
27:	0	0	0	0	0	...	0	0	0	0
28:	0	0	0	0	0	...	0	0	0	0

## DSPSC6 (SUBSCT) - SUBROUTINE

PURPOSE: DISPLAYS THE TRUNK SIZE MATRICES FOR SATELLITE, TERRESTRIAL AND MILITARY-OWNED TERRESTRIAL MEDIA.

INPUTS: SUBSCT - Indicates which trunk size matrix is to be displayed, as shown below:

- 1 - Satellite
- 2 - Terrestrial
- 3 - Military

OUTPUTS: Trunk size matrices to user screen.

CALLED BY: DISPLY.

CALLS TO : DSPMTX.

LOCAL VARIABLES:

Character:

TITLE - Holds titles for all three media.

### MODULE LOGIC AND DESCRIPTION:

This routine calls DSPMTX to display the requested matrix and uses SUBSCT as the pointer to the relevant starting point in NTRUNK, the trunk size matrix. Shown below is an example display of the trunk size matrix for each media type.

SECTION 6 - TRUNK SIZE						SUBSECTION 1 - SATELLITE				
S/D	1	2	3	4	5	...	25	26	27	28
1:	0	0	0	0	0	...	0	0	0	0
2:	0	0	0	0	0	...	0	0	0	0
3:	0	0	0	0	0	...	0	0	0	0
...										
27:	0	0	0	0	0	...	0	0	0	0
28:	0	0	0	0	0	...	0	0	0	0

# SUBSECTION 2 - TERRESTRIAL

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	1056	840	1440	312	...	792	432	408	432
2:	1056	0	480	1320	312	...	768	336	384	456
3:	840	480	0	720	168	...	840	168	144	216
						.				
						.				
27:	408	384	144	672	384	...	432	216	0	112
28:	432	456	216	552	168	...	504	120	112	0

# SUBSECTION 3 - MILITARY

S/D	1	2	3	4	5	...	25	26	27	28
1:	0	0	0	0	0	...	0	0	0	0
2:	0	0	0	0	0	...	0	0	0	0
3:	0	0	0	0	0	...	0	0	0	0
						.				
						.				
27:	0	0	0	0	0	...	0	0	0	0
28:	0	0	0	0	0	...	0	0	0	0

## DSPSC7 - SUBROUTINE

PURPOSE:     DISPLAYS THE OFFERED LOAD MATRIX.

INPUTS:      None.

OUTPUTS:     Offered load matrix to user screen.

CALLED BY:   DISPLY.

CALLS TO :   DSPMTX.

### LOCAL VARIABLES:

Character:

    TITLE - Holds titles for all three media.

### MODULE LOGIC AND DESCRIPTION:

This subroutine calls DSPMTX to display the offered load matrix OFLOAD and indicates that the elements which make up the matrix are of type REAL by passing 'R' as the third parameter to DSPMTX.

Shown below is an example of the offered load matrix display.

#### SECTION 7 - OFFERED LOAD

S/D	1	2	3	4	...	25	26	27	28
1:	0.0	609.3	484.7	830.7	...	456.9	249.3	235.3	249.3
2:	609.3	0.0	276.9	761.6	...	443.1	193.8	221.6	263.1
3:	484.7	276.9	0.0	415.4	...	484.7	96.9	83.1	124.7
					.				
					.				
					.				
27:	235.3	221.6	83.1	387.8	...	249.3	124.7	0.0	110.7
28:	249.3	263.1	124.7	318.5	...	290.7	69.3	110.7	0.0

## DSPSC8 - SUBROUTINE

PURPOSE:     DISPLAYS ALL OF THE NODAL SWITCH INFORMATION.

INPUTS:      None.

OUTPUTS:     Nodal switch information to user screen.

CALLED BY:   DISPLY.

CALLS TO :   CLR\_SCREEN, BUFFER.

### LOCAL VARIABLES:

Integer:

I,J - Loop variables.

DUMMY2 - Dummy variable used in call to BUFFER.

Character:

HDR - Holds titles for all three media.

DUMMY1 - Dummy variable used in call to BUFFER.

### MODULE LOGIC AND DESCRIPTION:

This routine displays all of the nodal switch information for the following five areas:

- 1 - Maximum number of alternates
- 2 - Protocol
- 3 - Maximum number of groups
- 4 - Minimum number of groups
- 5 - Maximum number of channels.

BUFFER is called to allow the user to enter <RETURN> before continuing. An example of each of these displays is shown below:

#### SECTION 8 - SWITCH DETAILS

SWITCH NUMBER:	1	2	3	4	...	25	26	27	28
	MAX NO OF ALTERNATES PROTOCOL								
MAX NO OF ALTS:	8	8	8	8	...	8	8	8	8
PROTOCOL:	2	2	2	2	...	2	2	2	2
MAX NO OF GRPS:	90	0	900	9	...	9	0	90	0
MIN NO OF GRPS:	1	1	1	1	...	1	1	1	1
MAX NO OF CHNLS:	90000	90000	90000	90000	...	90000	90000	90000	90000
	HIT <RET> OR D TO DUMP								

## DSPSC9 (SUBSCT) - SUBROUTINE

PURPOSE:     DISPLAYS DAMAGE AND NON-PREFERENTIAL ROUTE DATA.

INPUTS:     SUBSCT - Indicates which area is to be displayed, as listed below (Integer):

1,2	User and switch damage
3,4,5	Trunk damage
6	Non-preferred routes.

OUTPUTS:    Damage and non-preferential route data.

CALLED BY:   DISPLY.

CALLS TO :   CLR\_SCREEN, DSPMTX, BUFFER.

### LOCAL VARIABLES:

Integer:

NOARR - Equivalent to the array NOUSER so that NOARR can be used in the place of NOUSER.

DEAD - Holds relevant damage and routing information.

I,J,K,L - Loop variables.

DUMMY2 - Dummy variable used in call to BUFFER.

Character:

MESS - Messages pertaining to subsections.

DUMMY1 - Dummy variable used in call to BUFFER.

TITLE - Titles for subsections.

UNDRLN - Underline.

### MODULE LOGIC AND DESCRIPTION:

For SUBSCT values of 1 or 2, NOARR is searched for any values of 1, indicating damage, and the damaged node number is stored in the array DEAD. At the end of the search, either DEAD is output, if damage was found, or a message indicating that "all are operational" is sent to the user. For trunk damage, the relevant array is passed to DSPMTX to be output to the user screen. For non-preferred routes, if any non-preferred nodes are found, they are stored in DEAD and output to the user. If non-preferred routes exist, a message declaring this fact is sent to the user.

An example of each of these subsection displays is as follows:

SECTION 9 - DAMAGE & PREFERRED ROUTING    USERLESS NODES

.ALL ARE OPERATIONAL

HIT <RET> OR D TO DUMP



SECTION 9 - DAMAGE & PREFERRED ROUTING DEAD SWITCHES

ALL ARE OPERATIONAL

HIT <RET> OR D TO DUMP

SECTION 9 - DAMAGE & PREFERRED ROUTING DAMAGE TO TRUNKS -  
SATELLITE

NO DAMAGE TO THESE TRUNKS

HIT <RET> OR D TO DUMP

SECTION 9 - DAMAGE & PREFERRED ROUTING DAMAGE TO TRUNKS -  
TERRESTRIAL

NO DAMAGE TO THESE TRUNKS

HIT <RET> OR D TO DUMP

SECTION 9 - DAMAGE & PREFERRED ROUTING DAMAGE TO TRUNKS -  
MILITARY

NO DAMAGE TO THESE TRUNKS

HIT <RET> OR D TO DUMP

SECTION 9 - DAMAGE & PREFERRED ROUTING NON-PREFERRED ROUTES

THERE ARE NO NON-PREFERRED ROUTES

HIT <RET> OR D TO DUMP

ERROR(EPTR, EVAL, TYPE, \*) - SUBROUTINE

PURPOSE:     OUTPUTS ERROR MESSAGE AND VALUE(S).

INPUTS:     EPTR - Error message index number (Integer).  
              EVAL - Contains value of error, which may be node number, link endpoints, or  
                    alternate route number (Real).  
              TYPE - Determines type (format) of error message (Integer).  
              \* - Alternate return specifier (Integer).

OUTPUTS:     Error message.

CALLED BY:   PRCESS, OUTDAT, GETFIL, GETNAM, INDATA, PSNFIL, CHKSC1, CHKSC2,  
              CHKSC3, CHKSC4, CHKSC5, CHKSC6, CHKSC7, CHKSC8, CHKSC9, XCHECK,  
              DSPSC3, AMNSC9, INPNOD, AMNSCA, MENU6, INPVAL, REPLY, BUFFER.

CALLS TO :   BUFFER.

LOCAL VARIABLES:

Character:

ELINE - 60-character buffer that stores the error message text.  
ERRMSS(62) - Error message array.  
TEXT(4) - Qualifier array for error messages.

Integer:

I - Counter used to determine length of error message. Also used as source node  
    number.  
IFAC - Stores maximum node number limit.  
J - Destination node number, if applicable.  
K - Alternate route number, if applicable.  
LEN - Number of characters in error message.  
VAL - Stores value of EVAL while error message type is determined.

Logical:

END - Set to .TRUE. when error message decoding is finished.  
ENDSTR - Set to .TRUE. when the end of the error message string is found.

#### MODULE LOGIC AND DESCRIPTION:

The variables ENDSTR and I are initialized to .FALSE. and 45, respectively. The length of the specific error message is determined. TYPE determines how to process the error value. If TYPE equals 0, then no error value exists and only the error message text is displayed. If TYPE equals 1, the error message refers to general parameters, such as GOS limit. The specified parameter is passed to ERROR via the parameter EVAL. This number is printed to the user screen along with the error message text. If TYPE equals 2, the error was generated by an erroneous node value, link endpoints, or alternate route numbers. ERROR determines which of these three by using modular arithmetic. The appropriate error message is then printed to the user screen.

Lastly, if the error was not 'illegal input', the error message is cleared in preparation for the next error. This is done by calling BUFFER and passing it dummy variables.

## FNLCHK (OK) - SUBROUTINE

PURPOSE: CHECKS ALL SECTIONS AND SUBSECTIONS TO ENSURE THAT THEY ARE VALID.

INPUTS: OK (Logical).

OUTPUTS: OK - Set to .TRUE. if all data is present and validated.

CALLED BY: CONTRL.

CALLS TO: INDATA, CHECK, BUFFER, XCHECK.

### LOCAL VARIABLES:

#### Character:

LINE - Used to hold user's response.

YES - Default value for LINE.

#### Integer:

SECT - Points to the particular section of data that is in use.

SUBSCT - Points to a subsection within SECT.

IFLAG - Pointer for the two logical arrays DATAIN and DATAOK which indicate whether data exists for the current subsection and whether it is valid.

#### Logical:

OK - Set to .TRUE. if all data is present and validated.

END - Set to .TRUE. if no data is found for a subsection.

ERR - Set to .TRUE. if data exists but has not been validated.

CHECKD - Set to .TRUE. when all subsections have been validated.

### MODULE LOGIC AND DESCRIPTION:

The first section of code deals with a selected file that is being amended. The four logical flags, OK, END, ERR, and CHECKD are all set to .FALSE.. FPTR is a pointer to the files that are used. If a file is being amended it is the first file opened and hence FPTR is set to one. It first checks that subsections exist for the present section and then loops through searching for any section or subsection that has not been dealt with yet. If it finds one, it calculates SECT and SUBSCT, calls INDATA to read the data and CHECK to validate it. CHECKD is set to .TRUE. when all subsections have been validated.

The next section checks for a complete file. If a section is found without data, END is set to .TRUE.. If data is found but is not valid, ERR is set to .TRUE.. If either of these flags is set to .TRUE., an appropriate message is output and the user is asked if the file is still to be saved. BUFFER is called to accept the user's response. The response is compared to YES. If this is the case, OK is set to .TRUE. so that the output file is created.

The final section calls XCHECK if all the data exists, i.e. END is still set to .FALSE.. If XCHECK finds problems, the error route is used.

## GETFIL (TYPE, \*) - SUBROUTINE

PURPOSE: VERIFIES THAT CURRENT INPUT FILE ALREADY EXISTS AND IS OPEN. OPENS A NEW FILE IF NECESSARY AND SETS UP ATTRIBUTES.

INPUTS: TYPE - Indicates the type of file being assessed (Character).  
\* - Alternate return specifier (Integer).

OUTPUTS: Updated input file.

CALLED BY: MENU1, MENU4

CALLS TO: GETNAM, ERROR, PSNFIL

### LOCAL VARIABLES:

Integer:

ENDPTR - Points to the last file stored.

CPTR - Holds I/O unit number for current file.

Logical:

NEWFIL - Set to .TRUE. if current file has not been previously accessed.

### MODULE LOGIC AND DESCRIPTION:

The file and channel pointers (ENDPTR and CPTR) are initialized to 0 and 20, respectively. GETNAM is called with the type of file (TYPE) passed, so that the correct prompt is displayed. FILNAM is also passed to hold the user chosen filename. The file name is compared to those already stored and if found the file pointer is set accordingly and the flag NEWFIL is set to .FALSE..

If the file has not been accessed previously a blank file is opened. An error message is output if this cannot be done and control is passed back to the calling module using the ARS. If the file is opened successfully, the name and its channel are stored in arrays FILE and CHAN, respectively. The file is searched for the number of subsections in each section, which is stored in FILSCT. PSNFIL is called to position the file at the first section and to read the number of nodes, which is stored in the array NODE. If the number of nodes (NNOD) for the current file is 0, NNOD is set to the value read. This only happens if the file just opened is the one requested by the user to be amended. After determining the number of nodes, the file is rewound, its position is recorded in the array FLSTAT, and the channel pointer is incremented. Execution is then returned to the calling routine.

## GETNAM (FILNAM, TYPE) - SUBROUTINE

PURPOSE:     PROMPTS THE USER TO ENTER A FILENAME.

INPUTS:       FILNAM - First 12 characters of user-entered file name (Character).  
              TYPE - Indicates the type of file being assessed - A, I, or O (Character).

OUTPUTS:     Name of file to be amended, read, or created.

CALLED BY:    GETFIL, OUTDAT.

CALLS TO:     BUFFER, ERROR.

### LOCAL VARIABLES:

Character:

FILE - User-entered file name.

Integer:

LEN - Number of characters in user-entered file name.

### MODULE LOGIC AND DESCRIPTION:

TYPE holds one of three characters, 'A', 'I', or 'O' indicating amending, input, or output, respectively. TYPE determines the prompt message to the user, as follows:

A - Displays prompt message, "Name of file to be amended"

I - Displays prompt message, "Name of file to be read"

O - Displays prompt message, "Name of file to be created."

BUFFER is called with the values of 4, FILE, and LEN to read the user's reply. The character '4' is passed so that BUFFER does not output a message of its own, FILE holds the user's response, and LEN holds the length of the reply. If LEN exceeds 7, ERROR is called and the user is prompted again. Otherwise, FILNAM is set to the first seven characters of FILE. If the file name is less than 7 characters, the remaining characters up to 7 are spaces.

## HEADER(MENU) - SUBROUTINE

PURPOSE:     INFORMS THE USER WHETHER CREATING OR AMENDING A FILE, THE LAST FILE ACCESSED AND WHICH SECTION OF DATA IS CURRENTLY BEING USED.

INPUTS:      MENU - Indicates which menu called HEADER (Integer).

OUTPUTS:     Header information.

CALLED BY:   MENU1, MENU2, MENU3, MENU4, MENU5, MENU6, MENU7, CONTRL.

CALLS TO :   CLR\_SCREEN.

### LOCAL VARIABLES:

#### Character:

- PART1 - First part of header information.
- PART2 - Second part of header data.
- PART3 - Third part of header data.
- TEXT1 - Amending a file.
- TEXT2 - Creating a file.
- TEXT3 - Last file accessed.
- TEXT4 - Current section.

#### Logical:

- P2 - Set to .TRUE. if a file is being accessed.
- P3 - Set to .TRUE. when a section of data is being accessed.

### MODULE LOGIC AND DESCRIPTION:

MENU1 calls HEADER to set up PART1. Subsequent calls to HEADER set up PART2 and PART3. Once these are set, the logical flags P2 and P3 are set. These are then used to output the relevant parts of the header information, such as "AMENDING FILE", "CREATING A NEW FILE", "LAST FILE ACCESSED", and "CURRENTLY IN SECTION".

## INDATA (SECT, SUBSCT, OK, \*) - SUBROUTINE

PURPOSE: Reads in a section or subsection of data from the input data file.

INPUTS: SECT - Section number (Integer).  
SUBSCT - Subsection number (Integer).  
OK - Set to .TRUE. if a set of data is read in correctly (Logical).  
\* - Alternate return specifier (Integer).

OUTPUTS: OK (Logical).

CALLED BY: PRCESS, FNLCHK, CHKSC9, AMNSC9

CALLS TO: PSNFIL, ERROR

### LOCAL VARIABLES:

#### Character:

LINE - Used to skip over header information.

#### Integer:

FILNOD - Number of nodes in file to be read.

FILPTR - File pointer.

IUNIT - I/O unit number of file to be read.

I, J, K, L

M, N, M1, M2

M3, M, L1, L2 - Loop counters and index pointers for array handling.

#### Real:

OLDPRP - Old scaling factor.

SCALE - Scaling factor to adjust OFLOAD.

### MODULE LOGIC AND DESCRIPTION:

The I/O unit variable IUNIT is initialized along with the number of nodes for the file. PSNFIL is called to position the file at the required subsection in the required section. If it cannot be found, an error message is sent and control passed back to the calling module. If the subsection is found, the data is read. Once the data has been read, the flag OK is set to .TRUE. and control is passed back to the calling module.

## INIT (END) - SUBROUTINE

PURPOSE: THIS SUBROUTINE INITIALIZES ALL THE INPUT DATA ITEMS AND CALLS THE FIRST MENU.

INPUTS: END (Logical).

OUTPUTS: Initializes common variables and arrays, and returns parameter END to the main program signalling termination of input data updates.

CALLED BY: INPFMT.

CALLS TO: MENU1.

### LOCAL VARIABLES:

Integer:

SECT - Acts as an array pointer for subsection in use.

Logical:

LSTSCT - Loop control variable.

### MODULE LOGIC AND DESCRIPTION:

This routine initializes the arrays to zero and sets up default values for common input variables. INIT uses the array NOSBST to define the number of subsections within each section and the array FLGPTR to point to the start of each section.

Finally, INIT calls MENU1 to display the first menu using the parameter END to signal the termination of updating input data.



## INPFMT - THE MAIN PROGRAM FOR INPUT DATA FILE HANDLER

**PURPOSE:**     ALLOWS THE USER TO CREATE NEW DATA FILES OR AMEND EXISTING ONES.

**INPUTS:**     Inputs are simulation parameters such as the number of nodes, routing criteria, and offered load.

**OUTPUTS:**    The newly created or amended data file that is formatted for input to the network model. The file name is specified by the user.

**CALLED BY:**   None.

**CALLS TO:**    INIT, CONTRL, OUTDAT, CLOSE.

**LOCAL V.ARIABLES:**

    Logical:

        END - Set to .TRUE. when user has finished creating or amending data files.

        QUIT - Set to .TRUE. if no output file is to be created.

### MODULE LOGIC AND DESCRIPTION:

INPFMT is the main driver program for the input data file handler section. It allows the user to create new data files or amend existing ones. INPFMT first calls the subroutine INIT to initialize variables and data arrays. The subroutine INIT calls MENU1 to display the first menu. The user is given the choice in MENU1 to do one of the following:

- .     create a new data file (presently disabled option);
- .     amend an existing one;
- .     exit.

If the user does not wish to terminate editing the file, then the subroutine CONTRL is called. CONTRL calls subroutine MENU3 which allows the user to amend any of the major areas of the output file. PRCESS is called by CONTRL to handle a specific section of data, either chosen by the user or next section in sequence. After completing all data edits, CONTRL calls FNLCHK to finish processing and validation. INPFMT then calls OUTDAT to write any validated section or subsection that has been updated to a named input file. GETNAM is used by OUTDAT to collect the name. The subroutine CLOSE then closes all files used during the current session. INPFMT makes a final call to INIT to amend another data file or exit the program.

# INPNOD(NODE, ISTAT) - SUBROUTINE

PURPOSE: READS POSITIVE/NEGATIVE NODE NUMBER AND CHECKS RANGE.

INPUTS: NODE - Node number (Integer).  
ISTAT - Set to 0, when exit is chosen (Integer).

OUTPUTS: ISTAT (Integer).

CALLED BY: AMNSC9.

CALLS TO : BUFFER, ERROR.

## LOCAL VARIABLES:

### Character:

EXIT - Holds 'EXIT'.  
LINE - Holds user's reply.

### Integer:

IERR - Hold's I/O status.  
LEN - Number of characters in user's response.

### Logical:

LREAD - Set to .FALSE. when node has successfully been read.

## MODULE LOGIC AND DESCRIPTION:

The routine first initializes the variables LREAD to .TRUE. and ISTAT to 1. BUFFER is then called to read the user's response into LINE. ERROR is called if the number of characters in the response is less than one or greater than 4. The response is compared to 'EXIT'. If there is a match, LREAD is set to .FALSE., ISTAT is set to 0, and control is passed back to the calling routine. Error checking is then performed on the node to ensure that it exists. If the node falls outside the acceptable node range, ERROR is called and the loop is repeated. If no errors are encountered, LREAD is set to .FALSE. and the routine is left.

## INPVAL(TYPE, REQNO, SD) - SUBROUTINE

**PURPOSE:** THIS ROUTINE HANDLES NUMERIC INPUT FROM THE USER, PROMPTING THE USER FOR A SECOND VALUE WHEN AND IF REQUIRED. IT CONVERTS CHARACTERS TO NUMERIC VALUES.

**INPUTS:** TYPE - Type of input (Integer).  
REQNO - Number of values required to be input (Integer).  
SD(2) - Array to hold input values (Logical).

**OUTPUTS:** None.

**CALLED BY:** DSPSC3, DSPMTX, AMNSC1, AMNSC3, AMNSC8, AMNSC9, AMNMTX, AMNSCA, MENU6.

**CALLS TO :** BUFFER, ERROR, CHRINT.

### LOCAL VARIABLES:

#### Character:

BUFF - Holds user's response.  
EXIT - 'EXIT'

#### Integer:

DESC - Pointer to prompt message.  
I - Pointer for BUFF.  
IVAL(2) - Equivalent to the array LVAL.  
J - Number of decimal places in real value.  
LEN - Length of user's response in BUFF.  
NO - Number of values input by user.  
NUM - Numeric value of character number.  
VAL - Cumulative of NUM values.

#### Logical:

END - Set to .TRUE. when all the buffer has been processed.  
LVAL(2) - Local storage for user responses.  
REM - Set to .TRUE. if a decimal point is encountered.

#### Real:

RVAL - Equivalent to the array LVAL, and stores real number value from BUFFER.

### MODULE LOGIC AND DESCRIPTION:

This routine converts a line of data read from the user to 1 or 2 numeric values as specified by REQNO.

The variable DESC is set equal to TYPE and passed as a parameter to BUFFER. REQNO can be negative indicating the user can skip over the current value by hitting <RETURN>. If this is the case, the negative of DESC is passed. Otherwise, DESC is set to 4 to indicate that no prompt message is to be output by BUFFER, but that a valid response is required.

A large loop is entered after local variables have been initialized. The loop is controlled by the variable END which is set to .TRUE. only when the user has input the required number of valid values. BUFFER is called to accept the user's response with BUFF containing the user input and LEN containing the number of characters in the response.

If TYPE equals 1, the user can enter 'E' to exit. This is tested first and if the user has chosen this option, both LEN and REQNO are set to zero so that no other section of code is entered and the routine can be left. Otherwise, the line of user input is processed character by character. A loop is entered using I as the pointer for BUFF. CHRINT is called to convert one character to its equivalent numeric (i.e. '9' to 9) and the result is put into NUM. If NUM is less than or equal to 9, the value is added to VAL which stores the numeric equivalent of BUFFER. If REM is .TRUE., the variable J is incremented to indicate the number of positions after the decimal point required.

If TYPE is 2 (indicating a REAL is expected), the buffer character is checked for a decimal point. If REM has already been set, an error is recorded since two decimals points have been located. Otherwise, REM is set to .TRUE.. RVAL is set equal to VAL, and VAL and J are initialized to zero.

If NUM is not a numeral or decimal, the buffer character is checked for a space or comma, indicating a delimiter between two values. If the test fails, an unrecognizable character has been found and ERROR is called. If the buffer character passes the test, ERROR is still called if a real number is expected (since only one real is ever asked for) if there are already two values processed, or if VAL equals 0. Otherwise, NO is incremented and the value is stored in IVAL. VAL is reset to 0, and a check is made on REQNO. If NO exceeds REQNO and not all of the buffer has been processed, ERROR is called.

Once the input buffer has been processed, and LEN is not equal to 0 (indicating EXIT), the current value is stored. If TYPE is 1, indicating integer, the value is stored in IVAL. If TYPE is 2, indicating a real value, the value is stored in RVAL. The next check is to ensure that if two values were required, then two values have been input by the user. The loop control variable, END, is set to .TRUE. if the requirement has been met, otherwise the loop is repeated.

Outside the loop, if two values were required they are validated. If they fail the test, the whole procedure is repeated.

Finally, when the required number of values have been entered, they are copied from local storage to the parameter variables.

## MENU1(END) - SUBROUTINE

PURPOSE: THIS DISPLAYS THE FIRST MENU, ALLOWING THE USER TO CREATE A NEW FILE, AMEND AN EXISTING FILE OR EXIT.

INPUTS: END - Set to .TRUE. if user requests to exit (Logical).

OUTPUTS: MENU1 to user screen.

CALLED BY: INIT.

CALLS TO : CLR\_SCREEN, REPLY, GETFIL, HEADER, MENU2.

### LOCAL VARIABLES:

#### Character:

TITLE - Title line.

UNDLN - Underline.

TEXT(3) - Options user can choose from.

#### Integer:

I - Loop variable.

IANS - Holds user's option choice.

MAX - Parameter for number of user options.

#### Logical:

AMEND - Set to .TRUE. if user requests to amend an existing file.

### MODULE LOGIC AND DESCRIPTION:

CLR\_SCREEN is called to clear the user's screen. The header and the menu are then displayed and REPLY is called to accept the user's response. An example of this menu screen is shown below:

#### INPUT PREPROCESSOR - FILE EDITOR

\*\*\*\*\*

1. CREATE A NEW DATA FILE
2. AMEND AN EXISTING DATA FILE
3. EXIT

The parameters passed to REPLY specify maximum number of options, MAX, type of response required ('2' indicates a prompt message to be output) and IANS, the user's response. If the user specifies amending a file, the flag AMEND is set to .TRUE. and GETFIL is called to get the file name. The first parameter 'A' indicates that the file is to be opened for amending. HEADER is called to initialize and set up header information to be displayed at the top of all subsequent menus.

If the user chooses to create a new file, MENU2 is called passing the parameter END as exit is an option in the second menu. If the user chooses to exit, option 3 is selected and END set to .TRUE.. On exit, execution is returned to the calling routine.

## MENU2(END) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO DECIDE IF THE DATA FILE TO BE CREATED IS USED FOR DESIGN OR ANALYSIS PURPOSES.

INPUTS:      END - Set to .TRUE. if user requests to exit (Logical).

OUTPUTS:     MENU2 to user screen.

CALLED BY:   MENU1.

CALLS TO :   HEADER, REPLY.

### LOCAL VARIABLES:

#### Character:

    TITLE - Title line.

    UNDLN - Underline.

    TEXT(3) - Options user can choose from.

#### Integer:

    I - Loop variable.

    IANS - Holds user's option choice.

    MAX - Parameter for number of user options.

#### Logical:

    ANLYSE - Set to .TRUE. if user requests to analyze an existing network.

### MODULE LOGIC AND DESCRIPTION:

    This routine initializes the logical flag ANLYSE to .FALSE. and calls HEADER to display header information and then displays the title and menu. An example of this menu screen is shown below:

#### EDITING OPTIONS

=====

1. DESIGN A NEW NETWORK
2. ANALYSIS OF AN EXISTING NETWORK
3. EXIT

REPLY is called to accept and validate the user's response. ANLYSE is set to .TRUE. if reply is '2' and END is set to .TRUE. if option 3 is chosen.

MENU3(IANS, \*) - SUBROUTINE

PURPOSE:     DISPLAYS ALL THE SECTION HEADERS.

INPUTS:       IANS - Allows user to indicate section of interest (Integer).  
              \* - Alternate return specifier (Integer).

OUTPUTS:     Section headers to user's screen.

CALLED BY:   CONTRL.

CALLS TO :   HEADER, REPLY.

LOCAL VARIABLES:

Integer:

    I - Loop variable.

    MAX - Maximum number of user options.

MODULE LOGIC AND DESCRIPTION:

    Calls HEADER to display header information, displays the menu and calls REPLY to accept the user's response. An example of menu screen 3 is shown below:

#### AMENDING FILE

1. GENERAL NETWORK DETAILS
2. NETWORK COSTS & KILOMETRIC DISTANCES
3. ROUTING TABLE
4. MINIMUM LINK SIZE CONSTRAINTS
5. LINK CONNECTIVITY CONSTRAINTS
6. LINK SIZE DETAILS
7. OFFERED LOAD
8. NODAL SWITCH CONSTRAINTS
9. DAMAGE AND NON-PREFERRED ROUTING

If the abort option is used in REPLY, the control goes back to the calling routine via the alternate return specifier.

#### MENU4(SECT, SUBSCT, OPT4, IANS, \*) - SUBROUTINE

PURPOSE:     ALLOWS THE USER TO INDICATE WHERE THE DATA TO BE HANDLED CAN BE FOUND - TO BE INPUT FROM THE TERMINAL, FROM A DATA FILE, OR USE DATA ALREADY READ IN.

INPUTS:     SECT - Section number (Integer).  
             SUBSCT - Subsection number (Integer).  
             OPT4 - Set to .FALSE. if data is to be used from an existing file (Logical).  
             IANS - Allows user to indicate section of interest (Integer).  
             \* - Alternate return specifier (Integer).

OUTPUTS:     Display menu for method of data entry.

CALLED BY:   PROCESS.

CALLS TO :   HEADER, REPLY, GETFIL.

#### LOCAL VARIABLES:

Character:

COSTXT(2) - Headers for network cost and distance subsections.  
DAMTXT(6) - Headers for damage subsections.  
SUBTXT(4) - Subsection headers for media types.  
TEXT(3) - Menu options for method of data entry.  
TITLE - Title line.  
UNDRLN - Underline.

Integer:

I, J - Loop counters and pointers.

#### MODULE LOGIC AND DESCRIPTION:

This routine calls HEADER for header information and displays the title with underline and subtitle, if necessary. It then displays the menu for the method of data entry. An example of this menu screen is shown below:

##### METHOD OF DATA ENTRY

=====

1. AMEND/DISPLAY/ACCEPT DATA
2. USE DATA FROM AN EXISTING FILE
3. USE DATA ALREADY READ IN

REPLY is called to accept the user's reply, which is stored in IANS. If the user chooses to abort, the alternate return specifier is used to return control to PROCESS, the calling routine. If the user requests that the data come from an existing file (option 2), GETFIL is called to handle it and OPT4 is set to .FALSE.. If an error occurs here, the alternate return specifier is used to go back to the start of the MENU4 routine.



## MENU5(IANS, \*) - SUBROUTINE

**PURPOSE:** DISPLAYS THE OPTIONS AVAILABLE TO THE USER ON THE HANDLING OF THE DATA OR ALLOWS THE USER TO RETURN TO THE PREVIOUS MENU.

**INPUTS:** IANS - Allows user to indicate section of interest (Integer).  
\* - Alternate return specifier (Integer).

**OUTPUTS:** Display menu for handling data options.

**CALLED BY:** PRCESS.

**CALLS TO :** HEADER, REPLY.

### LOCAL VARIABLES:

#### Character:

TEXT(4) - Menu options for method of data entry.  
TITLE - Title line.  
UNDRLN - Underline.

#### Integer:

I - Loop counter.  
MAX - Maximum number of options.

### MODULE LOGIC AND DESCRIPTION:

This routine calls HEADER for header information and then outputs the title with underline and the menu. An example of menu screen 5 is shown below:

```
SECTIONAL DATA OPTIONS
=====
1. AMEND CURRENT DATA SECTION
2. DISPLAY CURRENT DATA SECTION
3. ACCEPT CURRENT DATA SECTION
4. RETURN TO PREVIOUS MENU
```

REPLY is called to accept the user's option choice and stores it in IANS. Again, the alternate return specifier is used for the abort option. If the user chooses to return to the previous menu, IANS is set to 0.

## MENU6(IANS, IVEC, \*) - SUBROUTINE

**PURPOSE:** DISPLAYS THE OPTIONS THE USER HAS WHEN AMENDING MATRICES AND ALLOWS THE USER TO RETURN TO THE PREVIOUS MENU.

**INPUTS:** IANS - Allows user to indicate section of interest (Integer).  
IVEC - Node of interest (Integer).  
\* - Alternate return specifier (Integer).

**OUTPUTS:** Display of menu options for amending matrices.

**CALLED BY:** AMNMTX.

**CALLS TO :** HEADER, REPLY, INPVAL, ERROR.

### LOCAL VARIABLES:

#### Character:

TEXT(7) - Menu options for method of data entry.  
TITLE - Title line.  
UNDRLN - Underline.

#### Integer:

I - Loop counter.  
MAX - Maximum number of options.

### MODULE LOGIC AND DESCRIPTION:

This routine calls HEADER to display header section. The title with underline are then displayed along with the matrix handling menu. An example of menu screen 6 is shown below:

#### MATRIX HANDLING OPTIONS

=====

1. SET MATRIX TO ONE VALUE.
2. SET A SOURCE NODE TO 1 VALUE.
3. SET A DESTINATION NODE TO 1 VALUE.
4. SET A SOURCE NODE TO VARIOUS VALUES.
5. SET A DESTINATION NODE TO VARIOUS VALUES.
6. SET VARIOUS SOURCE-DESTINATION POINTS.
7. RETURN TO PREVIOUS MENU.

REPLY is called to accept the user's response and stores it in IANS. If the user chooses to return to the previous menu, IANS is set to 0. If IANS equals one of the first five options, a source or destination node number is requested. After the user enters a source or destination node, INPVAL is called to read the node value into IVEC. ERROR is called if an invalid node number is entered.

## MENU7(IANS, \*) - SUBROUTINE

PURPOSE:     DISPLAYS THE NODAL SWITCH OPTIONS AND AN EXIT OPTION.

INPUTS:     IANS - Allows user to indicate section of interest (Integer).  
             \* - Alternate return specifier (Integer).

OUTPUTS:     Displays menu of nodal switch options.

CALLED BY:   AMNSC8.

CALLS TO :   HEADER, REPLY.

### LOCAL VARIABLES:

Character:

TEXT(6) - Menu options for method of data entry.  
TITLE - Title line.  
UNDRLN - Underline.

Integer:

I - Loop counter.  
MAX - Maximum number of options.

### MODULE LOGIC AND DESCRIPTION:

This routine displays header information, the title with underline, and the menu with nodal switch options. An example of menu screen 7 is shown below:

```
                SWITCH OPTIONS
                =====
1.  MAXIMUM NUMBER OF ALTERNATES
2.  PROTOCOL
3.  MAXIMUM NUMBER OF GROUPS
4.  MINIMUM NUMBER OF GROUPS
5.  MAXIMUM NUMBER OF CHANNELS
6.  EXIT
```

REPLY is called to accept the user's response and sets IANS to 0 if the exit option is chosen.

# NODMGE - SUBROUTINE

PURPOSE:     DISABLES ALL DAMAGE ATTRIBUTES.

INPUTS:       None.

OUTPUTS:      None.

CALLED BY:    CONTRL.

CALLS TO :    None.

## LOCAL VARIABLES:

Integer:

I, J, K - Loop variables.

LF - Pointer for data.

## MODULE LOGIC AND DESCRIPTION:

The first section of code sets all the elements in the user and switch matrices to 1 to indicate no damage. The next section sets the trunk matrix to undamaged and also sets no preferential routing. Finally, the code sets the arrays DATAIN and DATAOK to .TRUE. to ensure that the data is seen as valid.

## OUTDAT - SUBROUTINE

PURPOSE:     OUTPUTS DATA FOR EACH EXISTING SECTION TO A FILE NAMED BY THE USER.

INPUTS:      None.

OUTPUTS:     ASCII data file which serves as input to the model segment.

CALLED BY:   INPFMT.

CALLS TO:    GETNAM, DATE, TIME, BUFFER, ERROR.

### LOCAL VARIABLES:

#### Character:

TEXT - Displays section headers.  
COSTXT - Displays subsection headers for section 2.  
SUBTXT - Displays subsection headers for sections 4 and 5.  
DAMTXT - Displays subsection headers for section 9.  
DESC - Displays data validation text.  
SCFTR - Displays scaling factor header.  
CDATE - Holds system date.  
CTIME - Holds system time.  
OUTFIL - Holds name of output file.  
DUMMY1 - Dummy variable used in a call to BUFFER.  
VERNUM - Version number of the software

#### Integer:

OUT - Output channel number.  
DUMMY2 - Dummy variable used in a call to BUFFER.  
TOTSCT - Total number of sections.  
VALSCT - Number of valid sections/subsections.  
COUNT - Counter.

#### Real:

RT1 - Offered load.

#### Logical:

OK - Set to .TRUE. if current section of data is to be written to data file.

### MODULE LOGIC AND DESCRIPTION:

This subroutine receives input data for the common block. The first section of code opens a user-named file if data exists for the first section. This is because of the variable NNOD (the number of nodes in the network) which is used to size the output arrays. If data does not exist for the first section, the section of code which writes out the general data information is skipped and the subroutine proceeds with the code that writes the network cost and kilometric distance arrays out to the data file. The logical flag OK is set to .TRUE. if the section has been validated and is to be written out to this file. The program calls GETNAM to prompt the user for a name of the output file. Two pieces of code,

CDATE and CTIME, are system-dependent. They obtain the date and time from the VMS operating system and will have to be altered if the software is ported to another system.

The next section of code calculates the number of subsections that have data, and of these how many have been validated. It also sets up an array with the number of subsections that contain data within each section. (At present the maximum number of subsections is 20). The name of the output file, version number of the software, time, date, number of subsections, the number of subsections in each section, and the number of valid subsections are output as header information to the output file. Preceding each section of data, a section header is output. Preceding each subsection is a subsection header and a line indicating whether the data has been validated. If there is only one subsection in a section, no subsection header is output to the data file.

The rest of the module outputs headers and data for each section and subsection that contain data. The variable LF is initialized at the start of each section to point to the first flag of that section. The variable I is used to print out the correct message indicating the validity of the data.

Finally a completion message is printed to the screen indicating if the file has been created and written successfully, provided the flag OK has been set to .TRUE.; otherwise an error message is sent to the user's screen.

## PROCESS (SECT,END,\*) - SUBROUTINE

**PURPOSE:**     ALLOWS THE USER TO AMEND, DISPLAY, AND VALIDATE A PARTICULAR SUBSECTION OF DATA.

**INPUTS:**     SECT - Points to a particular section of data that is in use (Integer).  
                END - Outer loop control variable. Set to .TRUE. when user requests to save data or when each subsection has been validated (Logical).  
                \* - Alternate return specifier (Integer).

**OUTPUTS:**     Revised data file.

**CALLED BY:**   CONTRL.

**CALLS TO:**    MENU4, INDATA, MENU5, AMENDR, ERROR, DISPLY, CHECK.

### LOCAL VARIABLES:

#### Integer:

SUBSCT - Points to a subsection within SECT.

IANS - Users response to menus.

LF - Used as a pointer for the two logical arrays DATAIN and DATAOK which indicate whether data exists for the current subsection and whether it is valid.

#### Logical:

EXIT - Inner loop control variable. Set to .TRUE. when user requests in MENU5 to return to previous menu or when data cannot be found for the current subsection.

### MODULE LOGIC AND DESCRIPTION:

There are two loops that make up the subroutine CONTRL. The outer loop is controlled by END and SUBSCT, and is repeated for each subsection or until the user requests 'SAVE' on MENU4. The logical array DATAOK indicates whether the data for the current subsection has been validated or not. DATAOK is initialized to .FALSE.. SUBSCT is set to zero, if a section has no subsections; otherwise, it is initialized to one. MENU4 is then called from the outer loop of PROCESS. The menu prompts the user to enter the location of the data for the current subsection. INDATA is then called to read data from the file specified, if the user does not request 'SAVE'. If the specified input file is not found, EXIT is set to .TRUE..

The inner loop is controlled by the EXIT parameter and the logical array DATAIN which indicates whether data exists for the current subsection. When either EXIT or DATAIN are .TRUE., the loop is exited. Within this loop, MENU5 is called with parameters IANS and an alternate error route. IANS is the users response as to what should be done with the current section; options being amend, display or validate. The loop is repeated until the validation is complete on the section. If the user tries to display or validate a section where no data exists, the subroutine ERROR is called to output an error message to the user's screen.

Once the inner loop is left and provided EXIT is not set to .TRUE., the subsection counter and the flag pointer are both incremented and the outer loop is repeated until the user exits.

## PSNFIL (REQSCT, REQSUB, \*) - SUBROUTINE

PURPOSE: POSITIONS THE FILE POINTER TO THE REQUESTED SECTION OR SUBSECTION OF DATA.

INPUTS: REQSCT - Required section number (Integer).  
REQSUB - Required subsection number (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: File pointer set to the requested section or subsection of data.

CALLED BY: GETFIL, INDATA

CALLS TO: ERROR, CHRINT [FUNCTION]

### LOCAL VARIABLES:

#### Character:

LINE - Used to skip over header information.

#### Integer:

FILPTR - File pointer.

SUBSCT - Subsection counter.

UN - I/O unit number.

#### Real:

OLDPRP - Old scaling factor.

SCALE - Scaling factor to adjust OFLOAD.

#### Logical:

OK - Set to .TRUE. if set of data is read in correctly.

### MODULE LOGIC AND DESCRIPTION:

The variable EVAL is set up so that the section and subsection number can be passed to the error handling routine.

The first section of code determines whether any data exists in the required section/subsection of the file. If these tests fail, ERROR is called and control returns to the calling module. If data does exist for the required subsection, the I/O unit and file pointers are set up. The file is rewound, if the required data has already been passed.

The next section of code searches through the data file for the required data. The flag END is used to control this loop, which repeatedly reads lines from the file until a section header is met. The file pointer FILPTR is then incremented and the section number is converted to an integer for comparison. If this integer matches the required section and if a subsection has to be found, an inner loop is entered. The inner loop repeatedly reads the data file until the required subsection header is found. Once the required subsection is found, the loops are exited with END set to .TRUE.. If the required section is found and it has no subsections, the outer loop is left with END set to .TRUE..



The final section of PSNFIL skips over the next header line (provided it is not section 7 where a scaling factor has to be read) and sets the file pointer for the file. This pointer is set to a negative value when the file is correctly positioned at the bottom of the section.

REPLY(NOOPT, TYPE, OPTNO, \*) - SUBROUTINE

PURPOSE:     VALIDATES USER'S RESPONSES TO MENUS.

INPUTS:       NOOPT - Maximum number of options (Integer).  
              TYPE - Pointer to PROMPT message (Integer).  
              OPTNO - Option user selects (Integer).  
              \* - Alternate return specifier (Integer).

OUTPUTS:     Validated user response.

CALLED BY:   MENU1, MENU2, MENU3, MENU4, MENU5, MENU6, MENU7.

CALLS TO :   BUFFER, ERROR.

LOCAL VARIABLES:

Character:

    VALID - Holds all valid user responses in a table.  
    BUFF - Holds user's response.  
    YES - Holds 'YES'.

Integer:

    I - Pointer to table held in VALID.  
    LEN - Number of characters in user's response held in BUFF.  
    MAX - Maximum value in VALID.

Logical:

    FOUND - Set to .TRUE. when user's response is found in the table.  
    VALID - Set to .TRUE. if an affirmative response is made to abort message.

MODULE LOGIC AND DESCRIPTION:

The look-up table, VALID, contains the 12 possible replies to all the menus. BUFFER is called to prompt the user to enter a response. The parameters passed to BUFFER include TYPE which is used to hold the user's choice to prompt message, BUFF holds the user's response, and LEN is the length of the reply. Provided the length of reply is less than 6 characters, a search is done on the table VALID. ERROR is called if the response length exceeds 5 characters, or if the reply cannot be found in the table. The restriction NOOPT is passed to REPLY and if the table pointer I exceeds this value, ERROR is called.

If the user's response is found in the table, FOUND is set to .TRUE. and the search loop is left. The abort and save options can only be selected if TYPE equals 3. If these options are selected when TYPE is 2, ERROR is called. If the abort option is selected, a message is output to verify the response. If ABORT is set to .TRUE., the alternate return is used.

## WARN(WPTR, WVAL, \*) - SUBROUTINE

**PURPOSE:**     OUTPUTS A WARNING MESSAGE AND ALLOWS USER TO ACCEPT OR REJECT THE ENTERED VALUE.

**INPUTS:**     WPTR - Index to warning message text array (Integer).  
              WVAL - Warning value, which can be a GOS value, link endpoints, node or alternate route (Real).  
              \* - Alternate return specifier (Integer).

**OUTPUTS:**     Warning message to user screen.

**CALLED BY:**   CHKSC1, CHKSC5, CHKSC7, CHKSC8.

**CALLS TO :**    BUFFER.

### LOCAL VARIABLES:

#### Character:

          ACCEPT - Character string set equal to "ACCEPT".  
          LINE - Buffer used to read user's response to accept warning message.  
          WRNMSS(10) - Array of warning message tests.

#### Integer:

          LEN - Length of user response read by BUFFER.

#### Logical:

          END - Flag to indicate completion of warning value decoding.

### MODULE LOGIC AND DESCRIPTION:

Firstly, the subroutine writes the appropriate warning message text based on the warning pointer (WPTR) passed. Next the input parameters are checked to see if the warning refers to an erroneous GOS. If that is the case, then the GOS is output to the user screen. Otherwise, the warning value, WVAL, must be decoded. From this value it is determined whether the warning resulted from an erroneous node, link, or alternate route. The appropriate value or values are then written to the user screen.

Secondly, the user is prompted to accept the erroneous value, if desired. The subroutine BUFFER is then called to read the user's response into LINE with length LEN. After the user's entry is verified, program execution is returned to the calling routine.

## XCHECK (\*) - SUBROUTINE

PURPOSE: EXECUTES CROSS CHECKS ON VALIDATED DATA.

INPUTS: \* - Alternate return specifier (Integer).

OUTPUTS: None.

CALLED BY: FNLCHK.

CALLS TO : ERROR.

### LOCAL VARIABLES:

#### Integer:

I,J,K,L,M - Loop counters and array pointers.

TSM - Matrix which holds total number of trunks for all switch pairs over all three media - satellite, military, and terrestrial.

IFAC - Factorizing value.

#### Logical:

NULL - Set to .TRUE. if the matrix TSM is all zero.

#### Real:

ERR - Error value.

### MODULE LOGIC AND DESCRIPTION:

The first section of code loops through each of the trunk size matrices summing the values for each switch pair in the matrix TSM. The logical flag NULL is set to .FALSE. if any of the values are non-zero. If NULL is still set to .TRUE. after the summation, ERROR is called.

The first check ensures that for each existing link, there must be at least one trunk for that link. If there is no trunk, ERROR is called. The second cross-check ensures that for each existing route there is at least one trunk. This is done by checking the routing table against the TSM. Again, if there is no trunk, ERROR is called. The third check ensures that the number of damaged trunks does not exceed the total number of trunks for that link.

## 4.2 NETWORK SIMULATION

The network simulation segment consists of 32 modules, including the main program NETWORK. A data flow diagram of this segment is included in Appendix A, and the detailed module descriptions are given below in alphabetical order.

## BF (L1,L2,ICLASS) - FUNCTION

PURPOSE: TO CALCULATE THE BLOCKING FACTOR FOR AN INDIVIDUAL LINK FROM THE OFFERED AND THE CARRIED TRAFFIC STATISTICS.

INPUTS: L1 - Source node of the link to be analyzed (Integer).  
L2 - Destination node of the link to be analyzed (Integer).  
ICLASS - Class of the traffic to be analyzed (Integer).

OUTPUTS: BF - Link blocking factor (Real).

CALLED BY: ISIZE, HIGOS, LOGOS.

CALLS TO: ERROR.

### LOCAL VARIABLES:

Real:

CARR - Carried traffic for the specified link.  
OFF - Offered traffic for the specified link.

### MODULE LOGIC AND DESCRIPTION:

This function calculates the blocking factor for a given link, which is used to generate the value of the arithmetic link blocking factor, LLMEAN, for the subroutine ISIZE.

The blocking factor calculated by BF is only for the link being studied. The link's relative position in the network is not assessed. Hence, the link blocking factor does not take into account any rollback traffic, as rollback traffic is calculated separately in subroutine CLEAR\_TRAFFIC.

The formula for the calculation the blocking factor is the following:

$$\text{BLOCKING FACTOR} = \frac{\text{GROSS OFFERED TRAFFIC} - \text{GROSS CARRIED TRAFFIC}}{\text{GROSS OFFERED TRAFFIC}}$$

The argument ICLASS indicates which class of traffic is to be considered for the given link. Possible traffic classes consist of the following:

ICLASS = 1	Terrestrial traffic
ICLASS = 2	Satellite traffic
ICLASS = 0	Satellite and terrestrial traffic combined.

NOTE: Only Traffic Class 1 (Terrestrial) is used in the present version of the software.

#### BLKDAT - COMMON DATA SECTION

PURPOSE:     INITIALIZES OUTPUT DEVICE NUMBERS.

INPUTS:      None.

OUTPUTS:     Initialized variables.

CALLED BY:   None.

CALLS TO:    None.

LOCAL VARIABLES: None.

#### MODULE LOGIC AND DESCRIPTION:

      This subroutine initializes the output device numbers for the QTCM network simulation segment. These values are listed as follows:

          DATA IN /10/  
          DATA OUT /11/  
          DATA LOG /12/

      See Appendix B for a listing of common data areas and variables.

# BLOCK (OM,OV,N,B,VB) - SUBROUTINE

PURPOSE: TO CALCULATE BLOCKING FACTOR AND VARIANCE BLOCKING FACTOR WHEN A PARCEL OF ABRITRARY PEAKEDNESS IS OFFERED TO A TRUNK GROUP.

INPUTS: OM - Mean offered traffic (Real).  
OV - Offered traffic variance (Real).  
N - Number of trunks in link (Integer).

OUTPUTS: B - Traffic mean blocking factor (Real).  
VB - Traffic variance blocking factor (Real).

CALLED BY: ISIZE, RESIZE.

CALLS TO: ERLNGB, ERLNGC, PEAK, SMOOTH.

## LOCAL VARIABLES:

### Integer:

NEQ - Equivalent number of trunks in random traffic link.

### Real:

BEQ - Blocking factor of Equivalent Random Traffic trunk group.  
CARR - Mean carried traffic.  
CARV - Variance of carried traffic.  
EQM - Equivalent mean traffic.  
OP - Overflow traffic peakedness.  
OVAR - Overflow traffic variance.  
OVFLO - Overflow traffic mean.  
PK - Peakedness of the offered traffic.

## MODULE LOGIC AND DESCRIPTION:

The subroutine calculates the blocking B and its variance VB when traffic OM with variance OV is offered to N trunks. The three possible traffic distributions are as follows:

- |                    |           |
|--------------------|-----------|
| 1. Poisson traffic | $OM = OV$ |
| 2. Peaked traffic  | $OM < OV$ |
| 3. Smooth traffic  | $OM > OV$ |

The peakedness (PK) of the traffic is defined to be  $OV/OM$ .

The first part of subroutine BLOCK performs checks for simple types of traffic loads. For example, if there is no offered traffic, then there is no blocking and B is set equal to 0. If there are no trunks, then there is total blocking and B is set equal to 1.0. If it is determined that the Poisson traffic distribution is applicable, then the equality of the mean and the variance is defined to be within 1 percent. Furthermore, if there are twice as many trunks as the mean offered traffic, then regardless of the peakedness there should be little blocking and hence, the Poisson distribution is used for efficiency. When the traffic distribution is Poisson, Erlang's formula can be used to determine blocking



factors, so subroutine ERLNGB or ERLNGC is called, as appropriate. These routines return the blocking factor B.

Erlang's formula does not determine the variance of the overflow traffic, therefore the Benes' formula is used to calculate variance of the carried traffic. Since the offered traffic variance is known, a variance blocking factor is easily calculated.

If the offered traffic distribution is determined to be peaked, then Wilkinson's Equivalent Random Trunk Method is used to describe the distribution of the offered traffic. The essence of this method is that any peaked traffic can be represented as the overflow resulting from the offering of the equivalent mean (EQM) Poisson traffic to NEQ trunks. Offering this equivalent mean traffic to NEQ + N trunks is the same as offering the peaked traffic parcel to N trunks. Using the Poisson traffic component in the analysis, the Erlang and Nyquist formulas may be applied. Full mathematical details of the method are given in a paper by Steven Katz. This methodology does not provide a means of calculating the carried variance on the N trunks.

After the appropriate traffic distribution is determined, subroutine PEAK is called to calculate the Equivalent Random Traffic, EQM, and equivalent number of trunks NEQ. These two values are input to ERLNGB, which calculates the overflow traffic for the composite group, which is equivalent to the overflow arising when the peaked parcel is offered to N trunks. Nyquist's formula is then used to determine the overflow peakedness, so that overflow traffic and blocking factor variances can be calculated. It is highly desirable that the blocking factor variance is calculated using the variance of the carried traffic rather than the overflow traffic variance, because the overflow variances for various network links are not additive.

For smooth traffic distributions, Katz's method is used. This is similar to Wilkinson's method, and works on the basis that any smooth traffic of mean OM and variance OV can be considered as the traffic carried on NEQ trunks, when EQM Poisson traffic is offered [Wilkinson uses the overflow]. This carried traffic is then offered to the N trunks of the real group.

The subroutine SMOOTH is called to calculate the EQM and the equivalent number of trunks NEQ. If the smooth parcel can be carried on fewer trunks than the real group (i.e. if NEQ is less than N), there is no further blocking by routing through a link of N trunks.

Otherwise, if the real group is smaller (i.e. N is less than BEQ), then blocking occurs. The subroutine SMOOTH includes a modification to the Erlang B and Erlang C formulas to calculate the unique values of EQM and NEQ which carry traffic of exactly mean value OM and variance value OV. In this case, NEQ is a non-integer number.

After obtaining the value of EQM, the mean overflow traffic that results from offering EQM to N trunks is determined and consequently the mean carried traffic on the N trunks is calculated. Although the Benes' formula is used for calculating the carried variance. The use of this formula introduces some inaccuracy. However, while the value of the offered mean, OM, is less than the EQM, the traffic distribution is closer to being smooth than random, so the amount of error is minimized.

It should be noted that for all simulations the variance blocking factor is constrained to lie between 0.001 and 0.999 to eliminate any computational rounding errors.

## CHCKLS - SUBROUTINE

PURPOSE: CHECKS FINAL LINK STATISTICS FOR ANOMALIES, AND GENERATING WARNING MESSAGES TO THE LOG FILE, IF NECESSARY.

INPUTS: None.

OUTPUTS: Warning messages to log file for link statistics anomalies.

CALLED BY: Network.

CALLS TO: None.

### LOCAL VARIABLES:

#### Integer:

I, J - Link endpoints.  
NLNKS - Number of trunks in the link.

#### Logical:

LHD - Set to .FALSE. after the warning message subheader is output.  
LHEAD - Set to .FALSE. after the warning message header is output.

#### Real:

CARR - Gross carried traffic over link.  
GROFF - Gross offered traffic over link.  
NCARR - Net carried traffic over link.

### MODULE LOGIC AND DESCRIPTION:

CHCKLS executes two checks on the final statistics for each link (i.e. connected switch pair) in the network. These checks are listed below:

- 1) Verify that the gross carried traffic does not exceed the link capacity
- 2) Verify that the net carried traffic does not exceed the offered traffic.

In performing each check, all network links are examined one at a time.

To execute the first check, a loop is entered where I and J go from 1 to the number of network nodes. For each link, the total number of trunks for all media is calculated as well as the total gross carried traffic for both classes. These values are stored in NLNKS and CARR, respectively. NLNKS and CARR are then compared and if the carried traffic exceeds the link capacity, a warning message containing the link endpoints, the gross carried traffic and the number of links is written to the log file.

The second check is performed in a similar manner, except that the net carried traffic is compared to the gross offered traffic for each network link. These values are stored in NCARR and GROFF, respectively. If NCARR is greater than GROFF, then a warning message containing the link endpoints, the net carried traffic and the gross offered traffic is written to the log file.

## CHMAX (IFLAG, JFLAG) - SUBROUTINE

**PURPOSE:** TO SEARCH AND IDENTIFY ALL NODES THAT HAVE EXCEEDED THE MAXIMUM ALLOWABLE NUMBER OF TRUNKS. IF SUCH A NODE IS FOUND, TRUNKS ARE PROGRESSIVELY DELETED UNTIL THE NUMBER OF TRUNKS IS LESS THAN THE MAXIMUM.

**INPUTS:** IFLAG - Adjustment flag (Integer).  
JFLAG - Impasse flag (Integer).

**OUTPUTS:** Information messages to the log file.

**CALLED BY:** ISIZE.

**CALLS TO:** None.

**LOCAL VARIABLES:**

Integer:

K, M, N - DO loop/array indices.  
L(3) - Media preference table that contains the order of preferences for trunk deletion.  
MAX - Size of largest trunk group.  
MHI - End of largest trunk group.  
NCHAN(200) - Array of the number of trunks connected to each network switch.  
NHI - End of largest trunk group.  
TOTRUNK - Total trunks of all 3 media types in link.

Logical:

DELETE - Flag indicating whether trunk deletion is necessary.  
END - Set to .TRUE. to signal that all possible deletions are complete.

## MODULE LOGIC AND DESCRIPTION:

CHMAX initializes the NCHAN array to 0, and then sums the number of trunks of all three media at each node. This sum is stored in NCHAN. The subroutine then searches all nodes to identify those nodes with greater than the maximum allowable number of trunks. When such a node is found, all outgoing links are checked to find which one(s) have more than the minimum number of trunks allowed. When a link is found to have exceeded the maximum, the subroutine deletes trunks one at a time, from the largest trunk group with media preference L(K). The media preference table, L(K), is a choice table of the transmission facility media for deletion priority, preferred in the following order: TERRESTRIAL, MILITARY, and SATELLITE. Trunk deletions continue until all links are within acceptable ranges. If no trunk sizes are greater than the allowable minimum, then no deletions occur and a message stating the minimum network trunk size is written to the log file. This process tends to equalize group sizes. Since traffic is not considered in this process, all nodes can be handled in a single call to CHMAX. If any route is found where the maximum number of trunks is exceeded, but a reduction of trunks is impossible, the impasse flag, JFLAG, is set. If any adjustment has taken place, the adjustment flag, IFLAG, is set and the node is retested to determine whether the number of trunks is within limits. This process is repeated until the number of trunks is less than the maximum or until an impasse is reached.

## CLEAR\_TRAFFIC (I,LNOW) - SUBROUTINE.

**PURPOSE:** HANDLES UNSERVICED TRAFFIC REMAINING AT A NODE AFTER ALL ALTERNATE ROUTES OUT OF THE NODE HAVE BEEN USED. THIS TRAFFIC IS ROLLED BACK DOWN THE ORIGINAL ROUTE TREE UNTIL EITHER A NODE WITH REROUTING CAPABILITY IS FOUND, OR UNTIL THE ORIGINATING NODE IS REACHED. THE ROLLED BACK TRAFFIC IS PLACED IN THE OVERFLOW QUEUE AT THE NODE TO WHICH IT HAS BEEN ROLLED BACK, SO THAT IT CAN AWAIT REROUTING. TRAFFIC STATISTICS FOR THE ROLLED BACK TRAFFIC ARE ALSO ACCUMULATED.

**INPUTS:** I - Origination node (Integer).  
LNOW - Current node position of the blocked traffic parcel in the routing tree (Integer).

**OUTPUTS:** Informational messages to the log file.

**CALLED BY:** LOAD.

**CALLS TO:** PURSAT, ERROR.

### LOCAL VARIABLES:

#### Integer:

II - Node to which the traffic parcel is to be rolled back.  
JJ - Node from which the traffic parcel was rolled back.  
LL - Pointer into the PHIST array.  
NNOW - Blocked node under analysis.

#### Real:

BM1 - Mean blocked traffic for class 1.  
BM2 - Mean blocked traffic for class 2.  
BV1 - Variance of class 1 blocked traffic.  
BV2 - Variance of class 2 blocked traffic.  
B1 - Mean blocked traffic for class 1 (temporary variable).  
V1 - Variance for mean class 1 blocked traffic. (temporary variable).

### MODULE LOGIC AND DESCRIPTION:

Subroutine CLEAR\_TR moves blocked traffic from the node where it is blocked to a node where it can be properly rerouted according to the network rerouting protocols. A complication occurs if the blocked traffic is class 2 (satellite) traffic, because when traffic is offered to a satellite link, it is redesignated to be class 1 traffic for use in the routing algorithms. If this converted traffic is blocked, it must be redefined as class 2 traffic before being rolled back. The handling of this conversion is discussed in more detail in the description of the subroutine LOAD.

The next portion of the CLEAR\_TRAFFIC algorithm deals with the node-by-node rerouting protocols. For the purposes of explanation, consider the following example:

A traffic parcel of mean value R (in Erlangs) is blocked by other traffic at node L. The overflow parcel of the traffic has mean value O. This value is stored in the OFLARR

array at node K, and is awaiting alternate routing. At the same time, there is another overflow parcel of mean traffic N at node I awaiting alternate routing. The LSTATA array contains the gross carried traffic, A, for link I-K, and M for link K-L.

Now, with the aforementioned conditions, if the node K is designated with the routing protocol number 1, then it can reroute any traffic rolled back to it. Therefore, the traffic which is blocked at node L, is rolled back to node K. The rollback value, R, is simply added to the existing overflow value, O, at the node K.

The gross carried traffic on the link K-L now needs to be reduced by the rolled back value, R, because this amount of traffic was never actually carried to the node L. This adjustment is completed by incrementing the roll-back accumulator by R to reflect the additional rollback traffic, and then deriving net carried traffic from the difference of the gross carried traffic and the rolled back traffic, M-R.

If the node K is designated with routing protocol 2, then the switch at node K cannot accept rolled back traffic for rerouting purposes. Therefore, the traffic must be rolled back further until either a node with routing protocol 1 or the originating node is encountered. This means the accumulated carried and rollback traffic statistics for both links K-L and I-K must be adjusted for the rollback R. The rollback value, R, is added to the existing overflow value at the node I. The overflow at node K remains zero, as it passes any overflow back to node I.

For links with a single media type, the rollback of traffic of any class is a straightforward process. The traffic which rolls back over a pure terrestrial link retains its class, and is simply rolled back with appropriate adjustments to accumulated node and link statistics. For a pure satellite link, the rollback traffic must be changed from class 1 to class 2 traffic, since class 2 traffic is the only traffic which may be offered to a pure satellite link.

However, for multi-media links, the process is more complicated. The traffic which is rolled back is comprised of two elements, listed as follows:

1. Class 1 offered traffic (terrestrial component of the link) remains Class 1.
2. Class 2 offered traffic (the satellite component of the link) is changed into Class 1 to be offered, and changed back to Class 2 if rolled back.

In order to divide the classes of rolled back traffic correctly, it is necessary to access the stored value of the class proportion factor, CPF, which is calculated in subroutine STUB during forward loading. CPF is the proportion of the exiting class 1 traffic to the originating class 1 traffic for a given node. The CPF for each node is stored within the OFLARR(II,3,1) array element, where the given link is specified by originating node II of the given link.

The source code for this subroutine uses nested IF-THEN-ELSE blocks to separate the different protocol and class cases described above. Each time traffic is rolled back over a link, irrespective of the composition of the media type, the link's rollback counter, LPAC, is incremented by 1.

## DMNTWK - SUBROUTINE

PURPOSE: TO INCORPORATE THE DAMAGE INFORMATION SPECIFIED BY THE ARRAYS IN COMMON BLOCK COM5 INTO THE REMAINDER OF THE NETWORK DATA.

INPUTS: None.

OUTPUTS: IDEST - Damaged user or switch numbers (Integer).  
NTRUNK - Updated trunk size matrix to the log file (Integer array).

CALLED BY: NETWORK (the main program)

CALLS TO: None.

### LOCAL VARIABLES:

#### Integer:

CNODE - current node under examination.  
I,J,K - Array pointers, indices.  
NDEST - Number of "destroyed" switches or users.

#### Logical:

LDAMA - Set to .TRUE. when users, switches, or trunks are damaged.  
LDMSH - Set to .TRUE. until the trunk damage subheaders are written to the log file.  
LDAMH - Set to .TRUE. until trunk damage header is written to the log file.

#### Character:

SUBHDR(3) - Headers for writing the three trunk types to the log file.

### MODULE LOGIC AND OPERATION:

The subroutine DMNTWK executes the function of applying damage to network users, switches, and trunks.

In "damaging" users, the array NOUSER is examined for each switch. If users at the examined switch are damaged, then the traffic offered to this switch is set to zero in the array OFLOAD. A list is created of the user switches that are damaged, and this list is written to the log file after damage application.

To "damage" switches, three steps are required:

1. set the offered traffic for each switch to zero
2. set all trunk sizes for the switch to zero
3. set the weighted cost of using the switch arbitrarily high.

These three steps ensure that no calls are routed through damaged switches.

The array NOSWIT is examined to determine whether each switch is marked for damage by the QTCM user. The three steps above are executed for each switch found to be damaged. A list is maintained of the damaged switches and written to the log file after application of damage is complete.

To apply damage to network trunks, the trunk sizes of user-selected trunks are reduced to zero in the matrix NTRNK. The number of trunks to decrement is determined by the number of trunks in array NOTRNK for each trunk type (terrestrial, satellite, or military). After all decrements are performed, a list is produced of damaged trunks, grouped under trunk type and indicating the number of trunks remaining. This list is then written to the log file.

EB2 (A,N,B) - SUBROUTINE

PURPOSE: TO COMPUTE THE ERLANG B BLOCKING PROBABILITY (B) WHEN "A" ERLANGS OF POISSON TRAFFIC ARE OFFERED TO N TRUNKS.

INPUTS: A - Offered traffic to the N trunks [in Erlangs] (Real).  
N - Number of trunks to which traffic is offered (Integer).

OUTPUTS: B - Erlang B blocking probability (Real).

CALLED BY: ERLNGC.

CALLS TO: None.

LOCAL VARIABLES:

Integer:

I - DO loop index.  
J - Counter.

Real:

B1 - Intermediate variable used in algorithm calculating B.  
SUM - Accumulator.

MODULE LOGIC AND DESCRIPTION:

Computes the blocking probabilities, Erlang B (blocked calls lost), for an Erlang with N trunks.  
(This routine computes Erlang B faster than the Subroutine ERLNGB.)



# ERLNGB (A,N,B) - SUBROUTINE

PURPOSE: TO COMPUTE THE ERLANG B BLOCKING PROBABILITY (B) WHEN "A" ERLANGS OF POISSON TRAFFIC ARE OFFERED TO N TRUNKS.

INPUTS: A - Offered traffic to the N trunks [in Erlangs] (Real).  
N - Number of trunks to which traffic is offered (Integer).

OUTPUTS: B - Erlang B blocking probability (Real).

CALLED BY: BLOCK.

CALLS TO: None.

## LOCAL VARIABLES:

Integer:

I - DO loop index.

Real:

B1 - Intermediate variable used in algorithm calculating B.

## MODULE LOGIC AND DESCRIPTION:

The blocking experienced by a telecommunications system when A Erlangs of Poisson Random Traffic is offered to N trunks is calculated by using the Erlang B formula.

The subroutine calculates this function in an iterative manner, using the following recursive relationship:

$$B_N = \frac{A * B_{N-1}}{N + (A * B_{N-1})}$$

The lower limit on the size of B to be considered significant is  $1.0^{-35}$ .

ERLNGC (A,N,B) - SUBROUTINE

PURPOSE: TO COMPUTE THE ERLANG C BLOCKING PROBABILITY (B) WHEN "A" ERLANGS OF POISSON TRAFFIC ARE OFFERED TO N TRUNKS.

INPUTS: A - Offered traffic to the N trunks [in Erlangs] (Real).  
N - Number of trunks to which traffic is offered (Integer).

OUTPUTS: B - Erlang C blocking probability (Real).

CALLED BY: BLOCK, ERLS.

CALLS TO: EB2.

LOCAL VARIABLES:

Real:

A\_OLD - Previously offered traffic to the N trunks [in Erlangs].

A\_NEW - New offered traffic to the N trunks [in Erlangs].

Logical:

IDONE - Set to .FALSE. when blocking probability has been calculated.

MODULE LOGIC AND DESCRIPTION:

Calculates the blocking probability, Erlang C (blocked calls queued), for an Erlang with N trunks.

ERLS (A,S,ESA) - SUBROUTINE

PURPOSE: USED IN THE SMOOTHING PROCESS TO CALCULATE BLOCKING FACTORS FOR A NON-INTEGRAL NUMBER OF TRUNKS.

INPUTS: A - Random traffic mean (Real).  
S - Non-integral number of trunks to which traffic is offered (Real).

OUTPUTS: ESA - Blocking probability (Real).

CALLED BY: SMOOTH.

CALLS TO: ERLNGC, ERLNGB.

LOCAL VARIABLES:

Integer:

NFACT - Converts the non-integral number of trunks to an integer value.

Logical:

IDUMP - Set to .TRUE. if number of trunks (S) has a specific format.

MODULE LOGIC AND DESCRIPTION:

Subroutine SMOOTH calls ERLS rather than ERLNGB or ERLNGC to calculate blocking factors because S represents a non-integral number of trunks. An in-depth analysis of this routine is not provided, because the network design portion of the QTCM is not currently used.

## ERROR - SUBROUTINE

PURPOSE: TO OUTPUT AN ERROR MESSAGE.

INPUTS: MPTR - Index to error message text (Integer).  
EVAL - Error value to be displayed (Real).  
TYPE - Has a value of zero if an ERROR value exists (Integer).  
\* - Alternate return specifier (Integer).

OUTPUTS: Error message written to LOG file.

CALLED BY: BF, PEEK, SMOOTH, LOAD, CLEAR\_TRAFFIC, SCOUT, OCC.

CALLS TO: None.

### LOCAL VARIABLES:

#### Character:

EMESS - Entire error message buffer, including ERRMSS, TEXT, and EVAL.  
ERRMSS(27) - Array of error message texts.  
TEXT - Message text for error value: ' - ERROR VALUE.'

#### Logical:

ENDSTR - Set to .TRUE. when the end of the error message text is found.

### MODULE LOGIC AND DESCRIPTION:

After initializing ENDSR and I, ERROR enters an IF-THEN loop to determine the length LEN of the required error message. When the end of the message text is found, ENDSR is set equal to .TRUE.. The error message ERRMSS, and, if appropriate, the error value EVAL are then written to the log file. All possible error messages are listed as follows:

```
ERRMSS(1) - 'HEADER>UNABLE TO OPEN RUN NUMBER FILE      '  
ERRMSS(2) - 'HEADER>UNABLE TO OPEN LOG FILE              '  
ERRMSS(3) - 'HEADER>UNABLE TO OPEN OUTPUT FILE           '  
ERRMSS(4) - 'HEADER>UNABLE TO OPEN INPUT FILE            '  
ERRMSS(5) - 'HEADER>INCOMPLETE DATA FILE                '  
ERRMSS(6) - 'ISIZE>BLOCKING FACTOR NOT BETWEEN 0 & 1     '  
ERRMSS(7) - 'ISIZE>VRNCE BLOCKING FACTOR NOT BETWEEN 0 & 1 '  
ERRMSS(8) - 'ISIZE>DESIGN IS CYCLING: BEST APPROACH IS -  '  
ERRMSS(9) - 'ISIZE>UNRESOLVABLE GRP MAX VIOLATION AT NODE '  
ERRMSS(10) - 'ISIZE>UNRESOLVABLE CHANNEL MAXIMUM VIOLATION '  
ERRMSS(11) - 'ISIZE>DISTRBTN CANT BE BROUGHT WITHIN BOUNDS '  
ERRMSS(12) - 'ISIZE>MEAN CANNOT BE BROUGHT WITHIN BOUNDS '  
ERRMSS(13) - 'SCOUT> ERROR IN DECISION FLAGS              '  
ERRMSS(14) - 'LOAD> ERROR IN CALL TO SCOUT(T)             '  
ERRMSS(15) - 'LOAD> ERROR IN CALL TO SCOUT(S)             '  
ERRMSS(16) - 'CLEAR> MEAN(TERRESTRIAL) LESS THAN 0       '  
ERRMSS(17) - 'CLEAR> VARIANCE(TERRESTRIAL) LESS THAN 0   '  
ERRMSS(18) - 'CLEAR> MEAN(SATELLITE) LESS THAN 0          '
```

```

ERRMSS(19) -'CLEAR> VARIANCE(SATELLITE) LESS THAN 0      '
ERRMSS(20) -'BF> ICLASS NOT BETWEEN 0 & 2                '
ERRMSS(21) -'OCC> ICLASS NOT BETWEEN 0 & 2                '
ERRMSS(22) -'GOS> ICLASS NOT BETWEEN 0 & 2                '
ERRMSS(23) -'PEAK> OFFERED VARIANCE < OFFERED MEAN        '
ERRMSS(24) -'SMOOTH> OFFERED VARIANCE > OFFERED MEAN      '
ERRMSS(25) -'SMOOTH>NUMBER OF TRUNKS = 0                   '
ERRMSS(26) -'CLEAR> MEAN(TERRESTRIAL) GREATER THAN 0      '
ERRMSS(27) -'CLEAR> VARIANCE(TERRESTRIAL) GREATER THAN 0  '

```

## GPMAX (MM,NN) - SUBROUTINE

**PURPOSE:** TO SEARCH THE NETWORK FOR NODES WHERE MORE THAN THE MAXIMUM PERMITTED NUMBER OF GROUPS ARE CONNECTED. IF ANY ARE FOUND, THE DELETABLE GROUP CARRYING THE LEAST AMOUNT OF TRAFFIC IS DELETED.

**INPUTS:** None.

**OUTPUTS:** GTCM - Gross carried traffic mean for both class 1 and 2 type traffic (Real).  
INDEX - Node where trunk group maximum is violated (Integer).  
MM - Origination point of trunk group deleted (Integer).  
NN - Destination point of trunk group deleted (Integer).

**CALLED BY:** ISIZE.

**CALLS TO:** None.

### LOCAL VARIABLES:

Integer:

KOUNT - Counter of switches with too many trunk groups.  
M - DO loop/array index.  
M1 - Endpoint of smallest traffic group which can be deleted; network-wide parameter.  
M2 - Switch with group maximum violations.  
N - DO loop/array index.  
NGROUP(200) - Array containing the number of groups connected to each network switch.  
N1 - Endpoint of smallest network-wide traffic group which may be deleted.  
N2 - Endpoint of smallest network-wide traffic group out of switch M2; the switch with the most group violations.

Real:

TRMIN - Minimum traffic flow to/from a node.  
TRMIN1 - Minimum traffic flow in all of the network.

### MODULE LOGIC AND DESCRIPTION:

This subroutine searches all nodes and identifies nodes where there are more trunk groups connected than the maximum allowed. If no such nodes exist, then the subroutine returns with the values of MM and NN set to zero. If at least one node is found, then certain other criteria determine which trunk group should be deleted. If there is a node with too many groups, but no deletions are allowed because of constraints, the program does nothing except return with the value of MM set to -1.

The subroutine then proceeds into a double loop. In the inner DO loop, an examination is conducted on each trunk group that exceeds the maximum from a given node, M. If the group is carrying less traffic than TRMIN (the previously determined minimum allowed traffic flow into/out of this node M) and deletion does not violate other criteria (i.e. the node has more than the minimum number of groups, and the link connectivity is not mandatory), then the group is flagged as a candidate for deletion by storing the endpoint values in M2 and N2.

The outer DO loop then compares the traffic carried on the candidate link M2-N2 with the overall network's minimum value of traffic, TRMIN1. If the value of the traffic carried on link M2-N2, the smallest candidate link so far encountered, is smaller than TRMIN1, then link M2-N2 is marked to be deleted.

When both loops are complete, the deletion may be made. If KOUNT is greater than or equal to 1 but M1 has not been changed from its initialization value of zero, then a violating switch has been found but no candidate link can be identified for deletion. GPMAX then sets MM and NN equal to 0 and returns after identifying the switch exceeding trunk group maximum in the log file.

Otherwise, the candidate group is deleted from the NTRUNK array and the end-points are returned to the calling routine.

## HEADER(\*) - SUBROUTINE

PURPOSE: TO OPEN ALL FILES AND ENSURE THAT THEY ARE DEFINED PROPERLY FOR USE BY THE NETWORK (MAIN) PROGRAM.

INPUTS: \* - Alternate return specifier (Integer).

OUTPUTS: File logicals for the input, output, and log files.

CALLED BY: NETWORK.

CALLS TO: ERROR, DATE, TIME.

### LOCAL VARIABLES:

#### Character:

CDATE - System date.  
CTIME - System time.  
INFILE - Input file name.  
LINE - Buffer for each line of text read in.  
LOGFILE - Name of log file ('LOGFILE').  
RUNNO - The run number. [No longer used.]  
RUNPGM - The program name. [No longer used.]  
VERNUM - Program version number.

#### Integer:

IOS - I/O status of the open statement (0 if successful).  
NOSCTS - Number of validated sections in the input file.  
OUTFIL - Name of the output file ('OUTFILE').  
RNO - Logical data unit for run file. [No longer used.]  
SVF - Logical data unit of temporary output file.  
TXT - Logical data unit of run description file.  
VALSCT - Number of data sections in input file which should be valid. [Currently set at 20.]

### MODULE LOGIC AND DESCRIPTION:

HEADER first sets up a temporary file NTWKSVF to store output data if NETWORK is accidentally aborted while running.

The log file is then opened and the time and date stamps are derived from the system clock and written to the log file. Then, HEADER opens the output data file and prompts the user for the name of the input file preprocessed by INPFMT. The user's response is read into INFILE, and the selected file is opened. The first line of the input file is read into LINE, and echoed into the log file along with the name of the output file name. The number of sections is then read into NOSCTS to see if the file is complete. This is determined by comparing NOSCTS to VALSCT. A description of the run is then written to the log file, if a text file is present. Lastly, the log file is appended with the program name and version numbers.



## HIGOS(M,N,BFMEAN,IFLAG) - SUBROUTINE

**PURPOSE:** TO IDENTIFY AND REDUCE THE LINK GRADE OF SERVICE (IF FOUND TO BE TOO HIGH) OF A NETWORK LINK BY INCREASING THE NUMBER OF TRUNKS AVAILABLE. THIS SUBROUTINE IS A NETWORK DESIGN, AND IS CURRENTLY NOT USED.

**INPUTS:** M - Origination node of the link (Integer).  
N - Destination node of the link (Integer).  
BFMEAN - Mean link blocking factor (Real).  
IFLAG - Return flag indicating what action was taken by HIGOS (Integer).

**OUTPUTS:** NTRUNK - Adjusted trunk sizes if necessary (Integer array).  
Appropriate log file messages.

**CALLED BY:** XHI.

**CALLS TO:** SCOUT, BF.

### LOCAL VARIABLES:

Integer:

ICLASS - Traffic class indicator for call to function BF.  
ISAT - Traffic class indicator (satellite, terrestrial, military) for call to SCOUT.  
K,L - DO loop/array indices.  
LNOW - Pointer into the path history array.  
N1 - Origination node of the link.  
N2 - Destination node of the link.  
NCHAN(200) - The number of channels terminating at each network node.  
NGROUP(200) - The number of groups terminating at each network node.

### MODULE LOGIC AND DESCRIPTION:

A link with endpoints M and N is identified as having a grade of service which is too high. The program attempts to decrease this grade of service by expanding the primary route from node M to node N by adding one LTYPE trunk to each link. This addition is conditional upon the following constraints:

1. The blocking factor of the link is higher than the network mean.
2. It is permissible to connect LTYPE trunks in the link.
3. Adding a trunk does not cause the switches at either end of the link to exceed their maximum number of channels.

The restriction that only trunk groups with blocking factor greater than the mean may be adjusted is an attempt to preserve stability when in design mode. Also, since the switch pair (M,N) has implicitly a higher than average end-to-end grade of service, a component link of lower than mean blocking factor is most unlikely to be causing the high end-to-end grade of service. An exception to this is where satellite links are being accepted. In this case, the program creates a new pure satellite link. Where satellite trunks (LTYPE = 1) are being added, new trunk groups may be created at the minimum size, but for other media new routes are not created by establishing new links. Group and channel maximum criteria may not be violated, connectivity must be allowed, and at least EMIN Erlangs of traffic must be offered between the nodes M and N if a satellite group is to be created.

Although it may be most desirable to create the satellite link anyway and satisfy the group or channel maximum violations by removing another link/channel elsewhere, the use of the LTYPE in the program does not allow this. The topological check routines always remove such a new satellite link to correct the network as they can adjust only LTYPE trunks. A more sophisticated form of LTYPE may be useful in the future.

NGROUP is used to count the groups terminating at each switch (when attempting to add a new satellite group). NCHAN is used to count the channels at each switch. SWITCH(N1,5) contains the maximum number of channels permitted at switch N1. GROUP (N1,N2,LTYPE+1) contains the mandatory connectivity of LTYPE trunks between the switches N1 and N2. If this is zero, then connectivity is forbidden.

The status flag, IFLAG, returns the following values to the calling routine for the given circumstances:

IFLAG = 0	No changes have been made in this run of the subroutine; i.e. no trunks could be added.
IFLAG = +1	At least one trunk group (link) has had a trunk added.
IFLAG = -1	A satellite group has been created.

## HUNT (DATUM,ICALL,REQ,BEST,IHUNT) - SUBROUTINE

PURPOSE: THIS ROUTINE ACCEPTS, ONE AT A TIME, A SERVICE OF DATA POINTS AND INDICATES IF A CYCLE HAS DEVELOPED, AND IF SO TO POINT OF CLOSEST APPROACH TO THE SPECIFIED VALUE.

INPUTS: DATUM - Data point (typically the mean point-to-point GOS for a design iteration) (Real).  
ICALL - Ordinal number of datum point (typically, the design iteration number) (Integer).  
REQ - Specified reference value (Real).

OUTPUTS: BEST - Point of closest approach (Real).  
IHUNT - Cycle indicator flag (Integer).

CALLED BY: ISIZE.

CALLS TO: None.

### LOCAL VARIABLES:

Integer:

IUPPER - Upper index of POINTS array.  
K, M, N - Do loop/array indices.

Real:

ABSMIN - Absolute minimum approach.

### MODULE LOGIC AND OPERATION

The routine looks for a recurrent cycle in connected pairs of switches. In this way, HUNT is used to detect cycling in the QTCM network design mode. This cycling can occur as various network parameters are adjusted during design.

The datum points are stored in the array POINTS. The new datum point DATUM and its position in the array ICALL are passed as arguments. (Note that the calling routine must adjust the value of ICALL for each call to HUNT). POINTS is stored in a common data area to ensure that the values are retained throughout the program's execution.

If less than four points are already in POINTS, the routine does nothing with them except store the data and return to the calling routine.

Thereafter, the existing points are all compared with the new datum value. If an exact match is found, the preceding points are compared. If they also match, the network GOS has been repeated, implying that the network configuration has been repeated.

If a cycle is detected, the flag IHUNT is set and the points between the matching points and the end of the data set are compared with the target value REQ (generally, the required grade of service (RGOS) to find the data point which is nearest to REQ in value. This nearest data point is returned to the calling routine as BEST.

If more than 200 calls are made on the routine, only the last 200 points are considered.

ISIZE(RGOS,DGOS,EGOS,ICON1,ICON2) - SUBROUTINE

PURPOSE: TO CONTROL DESIGN AND ANALYSIS ITERATIONS AND GENERATE THE MODEL OUTPUT.

INPUTS: RGOS - Required grade of service, GOS (Real).  
DGOS - Design limit on the network GOS (Real).  
EGOS - Grade of service distribution limit (Real).  
ICON1 - First criteria for construction of the routing table (Integer).  
ICON2 - Second criteria for construction of the routing table (Integer).

OUTPUTS: OUTFILE.DAT - File containing the network performance data for later reformatting by the program OUTFMT.

CALLED BY: NETWORK.

CALLS TO: BF, BLOCK, CHMAX, GPMAX, HUNT, LOAD, ROUTE, SMALL, XHI, XLO, RESIZE.

LOCAL VARIABLES:

Integer:

BFC, I, IRB, IST, IOT, J - Dummy arguments for statement functions VAL1 and VAL2.  
I, L, M, MM, N - DO/array loop indices.  
IFIVER - Status flag showing if a primary path has a length of 5 or more links.  
IFL, JFL, IFLG, IHUNT - Status flags.  
II - Status flag indicating that trunks are to be deleted from an oversized link.  
ISAT - DO loop/array index.  
J, K, M, N - Dummy arguments for the statement functions TOT, BOTH, and ALL.  
KOUNT - Counter for the analysis iterations.  
KOUNTD - Counter for the design iterations.  
MEDIA - Temporary variable used to track the media type of the trunk to be deleted.  
MHI - Source node of a link with the highest GOS.  
MLO - Source node of a link with the lowest GOS.  
NHI - Destination node of a link with the highest GOS.  
NLO - Destination node of a link with the lowest GOS.  
NRFLAG - Flag indicating whether to construct the routing table.  
NSAT - Number of satellite trunks in a link.  
NSWTC - Total number of switches in the network.  
NTER - Number of terrestrial trunks in a link.  
NTRKS - Number of terrestrial trunks in a link.  
OUTFLG - A status flag.

Logical:

LRSIZE - Flag used to indicate if the trunk resizing subroutine should be called.

Real:

BEST - Mean network point-to-point GOS that is closest to RGOS, as determined by the subroutine HUNT.  
BFAC - Link blocking factor for the link which is composed of pure terrestrial or pure satellite facilities.

BLOC - Blocking factor (a temporary variable).  
 BSAT - Blocking factor for satellite trunks.  
 BTER - Blocking factor for terrestrial trunks.  
 CARSUM - Square of the sum of the carried traffic that is used as a test on convergence.  
 CL2BF - Blocking factor for class 2 traffic.  
 CL2VBF - Variance of the blocking factor for class 2 traffic.  
 E1, E2 - Counters used to determine the distribution for the link GOS.  
 LINKS - Number of links in the network.  
 LLMEAN - The mean link GOS.  
 LLSQ - Sum of the squares of the link GOS.  
 LLSUM - Sum of the link GOS.  
 LLVAR - Variance of the link GOS.  
 OBLTOT - Blocked traffic during the last iteration.  
 OCSUM - Square of the sum of the carried traffic during the last iteration.  
 OLLM - Mean link GOS in the last iteration.  
 OM - Mean offered traffic on each link.  
 OM1, OM2, OM3 - Components of the mean offered traffic on each link.  
 OPPM - Mean end-to-end user GOS during the last iteration.  
 OV - Variance of the offered traffic on each link.  
 OV1, OV2, OV3 - Components of the variance of the offered traffic on each link.  
 PAIRS - Number of switch pairs with a non-zero offered traffic value.  
 PARCST - Minimum traffic parcel size (a temporary variable).  
 PPGOS - Point-to-point grade of service.  
 PPHI - Highest point-to-point GOS in the network.  
 PPLO - Lowest point-to-point GOS in the network.  
 PPMEAN - Mean point-to-point GOS for the network.  
 PPSQ - Sum of the squares of all point-to-point GOS in the network.  
 PPSUM - Sum of the point-to-point GOS in the network.  
 PPVAR - Variance of the point-to-point network GOS.  
 P1 - Total cost of all satellite and military trunks (a temporary variable).  
 SAT - Number of satellite trunks in a link.  
 SATPRP - Proportion of trunks in a link that are satellite trunks.  
 TER - Number of terrestrial trunks in a link.  
 TERPRP - Proportion of trunks in a link that are terrestrial trunks.  
 TGOS - Target grade of service for a design run.  
 VB - Variance of the network blocking factor.  
 VBSAT - Variance of the blocking factor for satellite trunks.  
 VBTER - Variance of the blocking factor for terrestrial trunks.

#### STATEMENT FUNCTIONS:

ALL(M,N,K) - Given a link M-N, from node M to node N, ALL provides the link statistics summarized for both classes of traffic and for both directions of traffic flow in the link.

BOTH(M,N,K,J) - Given a link M-N, from node M to node N, BOTH provides the link statistics summarized for class J in both directions of the traffic flow in the link.

TOT(M,N,K) - Given a link M-N, from node M to node N, TOT provides the link statistics summarized for both class 1 and class 2 traffic.

VAL1(I,J,RB,ST,OT,BFC) - Used to prorate part of the rollback traffic.

VAL2(I,J,RB,ST,OT,BFC) - Used to prorate part of the rollback traffic.

#### MODULE LOGIC AND DESCRIPTION:

This subroutine, ISIZE, is long, but its structure is simple. The loop that performs the network analysis is embedded in the loop that performs network design.

The call to ISIZE from NETWORK is made with the arguments RGOS, DGOS, EGOS, ICON1, and ICON2 that specify the overall performance criteria for the network being designed or analyzed. If RGOS is input with a zero value, the program computes the output of the full results after the first analysis run. Therefore, if the QTCM user wants to do an analysis run, the value of RGOS should be set to 0.0 with the input formatter. The values of ICON1 and ICON2 determine which routing criteria are to be used in the construction of the routing table.

If the model is performing a design iteration, (RGOS > 0.0), then the maximum number of iterations performed is specified by the design count variable, DSNCNT, contained in the common block COM4. The QTCM user can modify the value of DSNCNT in the input file. The subroutine then defines the statement functions - TOT, BOTH, ALL, VAL1, and VAL2. The statement function TOT adds the links statistics for both classes of traffic. The statement function BOTH adds the link statistics in both directions. The statement function ALL adds the link statistics for both traffic classes in both directions. The statement functions VAL1 and VAL2 are used to distribute correct proportions of different class traffic in the traffic parcels.

The QTCM assumes that satellite and military facilities available to any network are relatively fixed in size, and therefore only the terrestrial transmission facilities may be adjusted. Although the design section of the source code allows any single medium (terrestrial, satellite, or military), to be adjusted by use of the variable LTYPE, the best way to design a multi-media network has not yet been fully explored.

Next, the initial values of the link blocking factor (LBF) and the variance blocking factor (VBF) are read from the input file established by the input formatter, INPFMT. Both LBF and VBF have a default value of 0.2. This value has been found to be approximately consistent with an overall 5% grade of service, which is often found in previously designed and analyzed networks. A more accurate value for LBF and VBF would bring about quicker convergence, but this does not appear to be critical. In a network link, the variance blocking factor is always greater than the mean blocking factor, because the traffic is "smoothed out" when passing over a link. However, by equating LBF and VBF in the beginning of ISIZE, the network analysis model assumes a Poisson, random, traffic distribution to bring about the computations of the blocking values for the next iteration quicker. The iterative process moves toward the correct value even though the initial blocking factor value had to be adjusted.

Note that ISIZE also initializes all link statistics in the statistical accumulator array, LSTATA. At the end of an iteration, the array LSTATA is copied to the array LSTAT and the array LSTATA is set to zero.

The minimum parcel size parameter, PARC, is set to an initial value read from the input data file. Any traffic parcels found during program execution which have a size less than the value of PARC are not routed according to the routing procedures. It is assumed that these "little parcels" arrive at their destination node without failure. Furthermore, it is assumed that these "little parcels" do not use any additional link capacity. The only exception to this procedure is if the parcel is at its originating node, in which case the parcel is considered to be blocked. The value of PARC is systematically reduced in the program according to the proportion of the total offered traffic which receives this special treatment. The lower limit on the value of PARC is stored in the parameter TINY. The use of parameter TINY imposes restrictions on the program so that better QTCM execution performance can be obtained. The default value of variable TINY is 0.0002, and can be modified by the QTCM user with the input formatter. As PARC gets reduced, the number of parcels handled by Subroutine LOAD increases, and more computer time is spent on decreasing the net effect. With this approach, when the link blocking factors are least accurate, the loading process is almost totally an approximation process. When the link blocking factors converge to some value, however, the level of detail modeled by the program in the loading process is greatly increased and the results are much more accurate.

It should be noted that during the loading process for dual-media networks, class 2 traffic becomes class 1 traffic when it passes over a satellite link. The CPF parameter states what proportion of carried traffic arising from class 2 offered traffic becomes class 1 traffic when offered to the dual media link. Parameter CPF is calculated in Subroutine STUB, and stored in common array OFLARR(1,3,1). For example, on a pure terrestrial link, the factor is zero and on a pure satellite link, it is 1.0. For the purpose of initialization, any dual-media link is assumed to carry traffic which has been distributed evenly to the number of trunks in that link.

The analysis loop follows inside the design loop. NRFLAG is used as a control variable indicating whether a new routing table should be constructed. A new table is necessary only when network connectivity has been changed during a design analysis or when there is no routing table in the input file. It should be noted that a new routing table is not automatically constructed if trunk sizes or costs have been altered. So, if design runs are being conducted on a network with widely varying costs or capacity, it may be desirable to recalculate the routing table for each design simulation. Additional computer time directly proportional to the number of network nodes is required to construct a new routing table.

Once a routing table is set up by Subroutine ROUTE, the traffic is loaded into the network for each switch pair, and the amount of traffic carried on each link is recorded. At the end of the loading process, a check is made for convergence of the analysis procedure. The basic criterion for convergence is that the net carried traffic on the links should be "the same" between two consecutive iterations. This is tested by calculating "carried sum of squares", parameter CARSUM, where the difference between the current iteration and the previous iteration carried traffic for each link is squared and summed. Convergence is attained when CARSUM is less than the parameter STABLE's value which is set during the input formatting, and which has a default value of 0.0001.

The next part of the set-up for the network analysis loop is to calculate the link blocking factors based on the traffic carried at the end of the last iteration. This calculation is a fundamental part of the KATZ-iterative method. The source code for ISIZE branches into three sections according to the trunk group type. In all cases the traffic offered at each link endpoint is added, and symmetrical/bidirectional trunks are assumed.

For pure terrestrial links, the traffic offered to the link is the sum of the class 1 and class 2 offered traffic, and is offered to all trunks, as these are all terrestrial. The terrestrial blocking factors are

calculated using the Subroutine BLOCK. Since there are no satellite trunks in pure terrestrial link, the satellite blocking factors are set to 1.0.

For pure satellite links, the offered traffic is only in class 2, because class 1 traffic cannot be routed over satellite facilities. The class 2 traffic is offered to all satellite trunks. The class 2 traffic that is carried becomes class 1 traffic after completion of the route because it has now been across a satellite link. The satellite blocking factors are calculated by Subroutine BLOCK. The terrestrial blocking factors are set to 1.0, as there are no terrestrial trunks in a pure satellite link.

For a multi-media link, the class 2 traffic is offered to the two media in proportion to the number of trunks, and all class 1 traffic is offered to the terrestrial trunks only. No overflow from one medium to another is permitted.

The Subroutine BLOCK uses Erlang's formulas to calculate the blocking factor and its variance when traffic is offered to a group of trunks of specified size. The statistics which accumulated for each link are as follows:

- . Offered traffic mean and variance
- . Carried traffic mean and variance
- . Rollback traffic mean and variance.

These values are maintained in the statistical accumulator array LSTAT. Traffic statistics are stored by traffic type, class, and blocking factors are stored by medium type (1 means satellite, and 2 means terrestrial). When all these calculations have been made, the next iteration begins.

If the analysis has converged or been terminated, a statistics are computed and written in summary format to the output file. Following the output of the summary statistics, the subroutine ISIZE then considers the required design parameters as follows:

If the input value of RGOS is found to be 0.0 during processing, then control is transferred to the main output part of the code.

If the QTCM is in design mode, then the network, as it has just been analyzed, is compared with the input parameters to determine if an acceptable design has been produced. If the network is unacceptable, a change is made, and another analysis loop is performed.

Subroutine HUNT is then called to check whether the design process is repeating a particular network configuration. The design iterations continue until the nearest approach to RGOS, the required grade of service, stored in BEST, is reached. When this occurs, a message is output, and control passes to the output section of ISIZE.

Topological constraints on the network are checked next. Subroutine GPMAX searches for nodes with more than the permitted number of trunk groups. If such a node is found, then one and only one link is deleted. A message is output, the new routing table flag is set, and a new design iteration begins. If it is not possible to delete a link, then a message is output and control passes to the main program to print output. If all nodes are within allowed trunk group levels, the program passes to the next test.



The maximum number of channels per node is checked for next. Subroutine CHMAX is called to perform this check. If a violation is found, the appropriate message is output.

The minimum group size is checked by calling Subroutine SMALL. If a violation is found, the appropriate message is output.

If the program is not aborted, then the designed network meets all the topological constraints.

The next section of the code includes a call to the link resizing routine RESIZE. This subroutine reduces the number of terrestrial trunks in each link until the blocking factor exceeds the threshold value stored in TINYBF. If the value of TINYBF is zero, then RESIZE is not called. Before the RESIZE may be called, the following criteria must be satisfied:

- . Current design iteration is one.
- . Link blocking factor threshold value is greater than zero.
- . Primary routing only is in force.
- . There are only terrestrial trunks in the network.
- . Every link blocking factor in the network is less than the threshold value.

After link resizing is done, the network performance and configuration are considered. The first criterion is the mean grade of service of all switch pairs. This value, PPMEAN, is compared with RGOS. If the difference is less than DGOS, the mean is considered to be a satisfactory value, and the distribution about the mean is considered.

If PPMEAN is not acceptable, then the highest or lowest possible switch pair grade of service is adjusted depending on whether PPMEAN is higher or lower than RGOS. Either Subroutine XHI or Subroutine XLO is called to complete this adjustment. In order to ensure that the traffic distribution is not worsened while adjusting the mean grade of service, a cut-off limit for the grade of service equal to RGOS is set and stored in the target grade of service parameter, TGOS. If PPMEAN is too high, then only switch pairs with a grade of service greater than RGOS are adjusted. If a necessary grade-of-service adjustment cannot be made, then a message is printed out and control passes to the main output and the program terminates.

If the mean grade of service is acceptable but the traffic distribution is too wide, then the program does not terminate. The grade of service is more important than the distribution. Therefore, the distribution is adjusted in a way that moves the mean grade of service nearer to RGOS, if possible. If PPMEAN is higher than RGOS, the distribution is adjusted from the high side, even if the point furthest from PPMEAN is low. It should be noted that switch pairs with grades of service lying within EGOS of PPMEAN are not adjusted.

Subroutine ISIZE conducts a minimum of 5 iterations. The iteration process can, however, be stopped after the number of iterations completed exceeds the maximum number of allowed iterations parameter value, NOITS. When executing design simulations the analyst is urged to set NOITS as close to 5 as possible. This way any errors that occur in the initial design stages can be handled efficiently. The number of iterations may be increased for the analysis mode of the QTCM, yielding outputs with higher accuracy. The output file contains an error message if the program stops because of reaching the value of NOITS.

## LOAD - SUBROUTINE

PURPOSE: TO APPLY OFFERED TRAFFIC SYSTEMATICALLY TO THE NETWORK AND DETERMINE TRAFFIC DISTRIBUTION.

INPUTS: None.

OUTPUTS: LSTAT - Updated link traffic statistics array (Real).  
NSTAT - Updated node traffic statistics array (Real).  
PPSTAT - Updated point-to-point traffic statistics (Real).  
Log file messages, as necessary.

CALLED BY: ISIZE.

CALLS TO: CLEAR\_TRAFFIC, SCOUT, STUB, PURSAT, ERROR.

LOCAL VARIABLES:

Integer:

I - Originating node.  
IFLAG - Indicates blocked traffic at a tandem node in the routing protocol 2 loading.  
INDEX - DO loop/array index.  
J - Destination node.  
K - Node read from the PHIST array (a temporary variable).  
L - DO loop/array index.  
LL - A pointer to the path history array.  
LNOW - A pointer to the path history array.  
NBLOK - Blocked node in routing protocol 2 loading.

Real:

SPARM1 - Subparcel mean value for class 1 traffic.  
SPARM2 - Subparcel mean value for class 2 traffic.  
SPARV1 - Subparcel variance value for class 1 traffic.  
SPARV2 - Subparcel variance value for class 2 traffic.

## MODULE LOGIC AND OPERATION:

LOAD first initializes the offered load and path history arrays. The current version of LOAD assumes that the offered load is both POISSON in its distribution and of class 2, i.e. has not been passed over a satellite link yet. Therefore, the mean and variance of the overflow class 2 traffic are set equal.

With multi-media networks, there are two classes of traffic to be routed at each node. The routes for these classes may not be identical, as class 1 traffic cannot be routed over a pure satellite link. If a link contains a mix of media, the class 1 traffic only uses the terrestrial trunks. However, any route that can be used by class 1 traffic can be used by class 2 traffic as well. Therefore, subroutine LOAD loads traffic only along class 2 routes which are determined by subroutine SCOUT with the parameter ISAT set to 2. Class 1 traffic is loaded along the class 2 route only if the routes for the next link in the path for both classes is the same. If the routes diverge, then the class 1 traffic is left in OFLAKR until the class 2 route coincides.

After initialization, a loop is entered to load the traffic offered onto the network. Inside of this loop, loading proceeds along the class 2 route, and at each step, the next link of the class 1 route is compared with the next link in the class 2 route. If these are found to be the same, then SPARM1 and SPARV1 are set equal to the overflow parcel mean and variance, respectively, for class 1 traffic. If there is no class 1 route or if the next links are different, then SPARM1 and SPARV1 are set equal to zero. The class 2 parcel for routing, with mean and variance values, SPARM2 and SPARV2, respectively, are always equal to the respective values in the class 2 overflow parcel. Whenever there is a parcel to be routed which is greater than the initial minimum traffic parcel size, PARC, then subroutine STUB is called to calculate the statistics for carried and overflow traffic. The loop then continues.

If neither SPARM1 or SPARM2 is greater than PARC, then the overflow arrays are checked. This is necessary because SPARM1 could have a zero value due to routing considerations even though a large class 1 parcel is awaiting routing. If the overflow parcels are both small, then they are not routed normally, but receive special handling. The accumulator parameter APLOST is incremented by the amount of the traffic receiving special handling. Then, if the parcels are at the originating node (LL is equal to 1) the traffic is considered to be blocked. If the parcels are not at the originating node then the parcels are considered to arrive at the destination. Node and point-to-point statistics are incremented to reflect this. The parcels are not, however, deemed to use up any further trunk capacity.

The small parcels are passed directly to the destination, instead of assuming them lost, because it is assumed that there are some network switch pairs with quite small offered traffic. Parcels between these switches can fall below PARC even on the primary route. If the small parcels were assumed lost, the final model output would then show no arriving traffic, or 100 percent blocking between the switches, when this is not the case. Whether the further loading of these small parcels would contribute to the network link statistics and thus justify the additional computation time is a function of the size of PARC compared with the mean offered traffic in the network. The accumulator ALPOST provides a reference to allow the QTCM user to adjust TINY, the minimum size of PARC, to optimize this balance between computation complexity and accuracy.

The next parcel is then loaded along the designated path until it becomes blocked. When a parcel is blocked at a node (i.e. no more permissible paths are available to the destination), the traffic handling routine CLEAR\_TRAFFIC is called. This routine rolls the blocked traffic back along its path until it reaches a node with protocol equal to 1, so that the parcel can be rerouted. Protocol 1 moves the blocked traffic back to the preceding node and adds it to the overflow traffic already there. Protocol 2 moves the blocked traffic parcel back to the origination, or the first node that has routing protocol 1, and adds it to the overflow traffic there.

Carried traffic link statistics accumulated by the QTCM are adjusted because the rollback traffic was not really carried on the link, but was counted as carried when the link was originally loaded. This process of rolling back traffic which the link could carry, but which was subsequently blocked, is unique to the algorithm used by the QTCM to achieve a stable traffic distribution. CLEAR\_TRAFFIC performs this correction. It should be noted that the call to CLEAR\_TRAFFIC is made only if there is significant residual traffic - the sum of the class 1 and class 2 traffic must be greater than  $1.0E-8$ . Otherwise, the residual traffic is considered to be insignificant and is assumed to be the result of approximation error. The program then steps back one node along the route by decrementing LNOW by one and looks for the next alternate route.

When the all node pairs have been loaded and all overflow traffic that can be rerouted has been rerouted, Subroutine LOAD returns control to the calling routine, ISIZE.

# LOGOS(M,N,BFMEAN,IFLAG) - SUBROUTINE

PURPOSE: TO INCREASE THE GRADE OF SERVICE (GOS) OF A SPECIFIED SWITCH PAIR (M,N) BY REDUCING THE NUMBER OF TRUNKS, IF POSSIBLE.

INPUTS: M - Source node (Integer).  
N - Destination node (Integer).  
BFMEAN - Mean link blocking factor (Real).  
IFLAG - Status flag for return (Integer).

OUTPUTS: NTRUNK - Adjusted trunk sizes if necessary (Integer array).  
Appropriate log file messages.

CALLED BY: XLO.

CALLS TO: SCOUT, BF.

## LOCAL VARIABLES:

Integer:

ICLASS - Traffic class indicator for call to BF.  
ISAT - Traffic class indicator for call to SCOUT.  
K, L - DO loop/array index.  
LNOW - Pointer into the path history array.  
N1 - Source node of link.  
N2 - Destination of link.  
NGROUP(25) - Number of groups terminating at the Ith node.

## MODULE LOGIC AND DESCRIPTION:

A switch pair (M,N) is identified to the subroutine as having a grade of service which is too low. The program attempts to increase this grade of service by deleting one trunk from each link on the primary route from M to N. However, removing a trunk may result in the removal of all LTYPE trunks or the reduction of the group size below the specified minimum value. These two cases are described as follows:

- (1) If removing a trunk would remove all LTYPE trunks, then this is permissible only if there is no mandatory requirement for LTYPE connectivity.
- (2) If removing a trunk would bring the trunk group size below the minimum value, then the group is deleted if it is comprised of LTYPE trunks, and the link blocking factor is below the average value for the network, and it is permissible to delete the group (in terms of the mandatory connectivity and minimum number of groups per switch). Otherwise, the group cannot be removed.

Once Subroutine LOGOS has reduced trunk sizes for switch pair (M, N), as permitted, the value of IFLAG is set and the routine ends. The status flag, IFLAG, returns the following values:

IFLAG = 0 No changes have been made during this iteration.  
IFLAG = +1 At least one trunk has been deleted.  
IFLAG = -1 A group has been deleted.

## NETWORK - MAIN PROGRAM

PURPOSE: TO READ IN DATA FROM THE FORMATTED INPUT FILE AND INITIATES A PROGRAM RUN.

INPUTS: Formatted input data file - User's choice.

OUTPUTS: Output data file OUTFILE.DAT - unformatted.  
Log file LOGFILE.DAT - formatted.

CALLED BY: None. NETWORK is the main program.

CALLS TO: DMNTWK, ISIZE, HEADER, CHCKLS.

### LOCAL VARIABLES:

#### Character:

LINE(80) - Buffer used to read input file line by line.

#### Logical:

START - Set to .TRUE. when the start of the input file has been encountered.

#### Integer:

ICON1 - First minimization parameter for creating a routing table.

ICON2 - Second minimization parameter for creating a routing table.

IX - Null routing table indicator.

L, M, M1, M2, M3, M4, N - DO loop/array indices.

NOSCTS - Number of data sections in the input file; determines whether or not the input file was validated by the input formatter.

#### Real:

DGOS - Design tolerance of the computed mean point-to-point grade of service with respect to the required grade of service, RGOS.

EGOS - Grade of service distribution limit.

PROP - Traffic load proportion factor.

RGOS - Required mean point-to-point GOS.

WTGMIL - Weighting factor for the cost of military trunks (referenced to terrestrial trunk costs).

WTGSAT - Weighting factor for the cost of satellite trunks (referenced to the terrestrial trunk costs).

### MODULE LOGIC AND DESCRIPTION:

After initialization, NETWORK prompts the user to choose Erlang B or Erlang C handling of blocked calls. The program then begins reading the input data file that the QTCM user preprocessed with the input data formatter. Lines of data are read from the file until the word 'SECTION' is located. The variable START is then set to .TRUE.. The input data are assumed to have been validated by the input formatter, INPFMT. The number of valid data sections in the file is read from the input file. If the file does not have this number of data sections, then an error message is written to the output file and the network simulation segment of the QTCM stops.

If the data file is complete, then the program skips over all comment statements in the data file until the first section of the data is encountered. The data are then read in from the input file and written out to the output file one section at a time.

When Section 3 for routing is reached, a flag called IX is read in from the input file to indicate whether or not a routing table is to be read in or not. If its value equals zero, the first element of the routing table array, NROUTE(1,1,1,1), is set to 99 to indicate that a routing table must be created. Also, when the NETWORK program reaches Section 7 for offered load, the offered load for all network links is accumulated in ITOTOL for output to the screen.

When all data have been read in, then NETWORK calls DMNTWK to simulate network damage as defined in Section 9 of the input file. [Note that if the immediate effect of the damage on a given network is to be analyzed, then a routing table must be input. Otherwise, a new and optimized routing table is created based on the damaged network.] Once the network damage simulation is complete, the program NETWORK calls ISIZE. ISIZE is the principal controlling subroutine of the QTCM network simulation. If ISIZE completes normally, the program NETWORK is reentered and calls CHKCLS to check the link statistics for the run. If ISIZE does not complete normally, an error message is written to the log file and the program NETWORK terminates.

# NEXTHI(M,N,RGOS,IFLAG) - SUBROUTINE

PURPOSE: FIND THE NETWORK SWITCH PAIR WITH THE HIGHEST GRADE OF SERVICE (GOS) WHICH IS BOTH LOWER THAN THE GOS FOR A GIVEN SWITCH PAIR (M,N) AND HIGHER THAN THE REQUIRED GRADE OF SERVICE VALUE, RGOS.

INPUTS: M - Source node (Integer).  
N - Destination node (Integer).  
RGOS - Required minimum grade of service (Real).  
IFLAG - Return status flag (Integer).

OUTPUTS: M,N - Endpoints of new user pair (Integers).

CALLED BY: XHI.

CALLS TO: None.

## LOCAL VARIABLES:

Integer:

MM, NN - DO loop/array indices.  
M1, N1 - Endpoints of current switch pair.

Real:

PPGOS - Point-to-point GOS.  
PPLOW - Point-to-point GOS of switch pair M1, N1.  
PPMAX - Point-to-point grade of service of the original switch pair M, N.

## MODULE LOGIC AND DESCRIPTION:

This subroutine searches for and returns the values of the network switch pair with the highest GOS which is both lower than the GOS of the input pair (M,N) and higher than the required minimum grade of service, RGOS. If such a pair is found, then the subroutine returns with IFLAG set to 1, and values of M and N set equal to the values of the new switch pair. If no pair can be found, then IFLAG is set to zero, and the original input values remain in M and N.

Point-to-point grades of service are calculated directly from carried and offered traffic. The routine steps through all switch pairs, finding the highest GOS pair meeting the constraints and replacing it if a pair with higher GOS is found.

# NEXTLO(M,N,RGOS,IFLAG) - SUBROUTINE

PURPOSE: TO FIND THE NETWORK SWITCH PAIR WITH THE LOWEST GRADE OF SERVICE (GOS) WHICH IS BETWEEN THE GOS FOR A GIVEN END-TO-END USER PAIR (M,N) AND THE REQUIRED GRADE OF SERVICE VALUE, RGOS.

INPUTS: M - Source node (Integer).  
N - Destination node (Integer).  
RGOS - Required grade of service value (Real).  
IFLAG - Return status flag (Integer).

OUTPUTS: M, N - Endpoints of new user pair (Integers).  
IFLAG - Return status flag.

CALLED BY: XLO.

CALLS TO: None.

## LOCAL VARIABLES:

### Integer:

MM, NN - DO loop/array indices.  
M1, N1 - Link endpoints of current lowest GOS pair.

### Real:

PPGOS - Point-to-point GOS.  
PPLOW - Point-to-point GOS of pair M1, N1.  
PPMIN - Point-to-point GOS of the original pair M, N.

## MODULE LOGIC AND DESCRIPTION:

This subroutine returns the values of the switch pair with the lowest GOS which is higher than the GOS of the input pair (M,N) and lower than RGOS. If such a pair is found, then the subroutine returns with IFLAG set to 1 and values of M and N set to the values of the endpoints of the new pair. If no pair can be found, then IFLAG is set to zero, and the original input values remain in M and N.

Point-to-point grades of service are calculated directly from carried and offered traffic. The routine steps through all network switch pairs to find the lowest GOS pair meeting the constraints.



## PEAK(OM,OV,EQM,NEQ) - SUBROUTINE

PURPOSE: TO CALCULATE THE VALUES OF EQM AND NEQ SUCH THAT WHEN RANDOM TRAFFIC OF MEAN EQM IS OFFERED TO NEQ TRUNKS, THE OVERFLOWING PEAKED TRAFFIC HAS MEAN OM AND VARIANCE NEAR OV (I.E. WILKINSON'S EQUIVALENT RANDOM TRUNK THEORY APPROXIMATED BY THE USE OF INTEGRAL TRUNKS).

INPUTS: OM - Peaked traffic mean (Real).  
OV - Peaked traffic variance (Real).

OUTPUTS: EQM - Equivalent random traffic mean (Real).  
NEQ - Equivalent number of trunks (Integer).

CALLED BY: BLOCK.

CALLS TO: ERROR.

### LOCAL VARIABLES:

#### Integer:

K - DO loop/array index.  
NBEST - Best value of number of trunks in equivalent group.  
NRAPP - Value of RAPP.  
NSTAR - The number of trunks in equivalent groups.

#### Real:

A - Rapp approximation to equivalent mean.  
ABEST - Best value of offered traffic to equivalent group.  
ASTAR - Offered mean to equivalent group.  
DELTAV - Discrepancy in variance.  
DENO - Temporary variable denominator.  
MSTAR - Overflow mean from equivalent group.  
RAPP - Rapp approximation to equivalent number of trunks.  
SUM, TEMP - Calculation accumulators.  
VSTAR - Overflow variance from equivalent group.

### MODULE LOGIC AND DESCRIPTION:

This subroutine determines an offered Poisson traffic, EQM, which yields an overflow traffic of mean value OM and variance value of OV. The offered Poisson traffic determined by this subroutine is further constrained in the sense that it is offered to NEQ trunks in the system, where NEQ is calculated by the subroutine as well.

If non-integral trunk group sizes are allowed, an exact and unique solution can be found for any pair (OM,OV). However, the complexity of this process would require nested iterative solutions, which are process-intensive. Therefore, the present version of the code introduces marginal error by restricting itself to using integral trunks to economize on the computer processing time.

Using integral trunks, either the overflow mean or the overflow variance, but not both, can be made to approach the desired value with arbitrary accuracy. The approach is taken to make the means OM and MSTAR agree, and get the best agreement possible between the mean and peak variances, VSTAR and OV.

The method used in PEAK is essentially trial-and-error. In principle, the initial estimate of the number of trunks can start at 1 and increase incrementally. However, the use of Rapp's approximation offers an advantage in the initialization of the iterations. For the range of the means and the variances commonly encountered within a telecommunications network, Rapp's approximation has been shown to be inaccurate when used alone. Therefore, if Rapp's estimate is less than 10 equivalent trunks, the iterations are started at 1. Otherwise, the iterations are started at the Rapp's estimated value minus 5.

Knowing the initial estimated number of trunks, NSTAR, Wilkinson gives a formula to estimate ASTAR, the offered traffic mean, as shown below:

$$\text{ASTAR} = \frac{\text{OV} * (\text{NSTAR} + \text{OM} + 1.0) + \text{OM}^2 - \text{OM}}{\text{OV} + \text{OM}}$$

where OM is the peak traffic mean  
OV is the variance of peak traffic mean

This estimated value of ASTAR is then offered to the NSTAR trunks and the overflow mean, MSTAR, is calculated using the Erlang B formula, which is coded inside Subroutine PEAK to save computation time.

Keeping the NSTAR trunks constant, the offered load (ASTAR) is adjusted until the overflow mean (MSTAR) approaches the desired value of OM to an arbitrary degree (in this case plus or minus 0.1). This value is set quite high to speed up the iterations. Because of the integral nature of the solution, the overall effect of this on overall run-time is negligible.

At this point we have ASTAR and NSTAR values which generate MSTAR equal to OM. Now we need to compare the overflow variance VSTAR with the target value OV.

Examination of the nature of the solutions has shown that if MSTAR is kept equal to OM as NSTAR is varied, VSTAR steadily approaches and then moves away from OV. This relationship is used to stop the iteration, when the difference between VSTAR and OV (DELTAV) increases.

NSTAR is gradually increased, while ASTAR is adjusted to make MSTAR equal to OM, and DELTAV is calculated. As long as DELTAV is decreasing, the ASTAR, NSTAR pair replace the best values (ABEST, NBEST) so far. As soon as DELTAV increases, the iteration terminates and ABEST and NBEST are output as EQM and NEQ.

PURSAT(M, N) - FUNCTION

PURPOSE: TO DETERMINE IF A GIVEN TRUNK GROUP IS PURE SATELLITE.

INPUTS:     M - Source node of link (Integer).  
          N - Destination node of link (Integer).

OUTPUTS:    PURSAT - Pure satellite flag (Logical).

CALLED BY:   ROUTE, SCOUT, LOAD, CLEAR\_TRAFFIC.

CALLS TO:    None.

LOCAL VARIABLES: None.

MODULE LOGIC AND DESCRIPTION:

The NTRUNK array is examined to determine the type of trunk groups between link endpoints M and N. If there are only satellite-type trunks, then PURSAT is set to .TRUE.. Otherwise, PURSAT is set to .FALSE.. The PURSAT function shortens and simplifies expressions in the calling routines - ROUTE, SCOUT, LOAD, and CLEAR\_TRAFFIC.

## RESIZE - SUBROUTINE

PURPOSE: TO ADJUST THE NUMBER OF TERRESTRIAL TRUNKS OF A NEW NETWORK TOPOLOGY SO THAT THE BLOCKING FACTOR IS JUST SLIGHTLY LESS THAN TINYBF.

INPUTS: LSTATA - Network statistic array (Real).  
NTRUNK - Trunk size array (Integer).  
GROUP - Mandatory link sizes array (Integer).  
TINYBF - Minimum blocking factor (Real).

OUTPUTS: NTRUNK - Reduced trunk size array (Integer).

CALLED BY: ISIZE.

CALLS TO: BLOCK.

LOCAL VARIABLES:

Integer:

IPTT - Number of terrestrial trunks in the current link.  
RTRNK - Number of trunks removed from the current link.  
NODEF - Source node for link.  
NODET - Destination node for link.  
MANDT - Minimum number of trunks permitted in the link.

Real:

BF - Link blocking factor.  
VBF - Variance blocking factor.  
GOM - Gross offered mean over the link.  
GOV - Gross offered variance over the link.

## MODULE LOGIC AND DESCRIPTION:

During the initial design of a network topology, it is assumed that links are oversized and that traffic is routed on the primary path only. However, once the network topology is determined in terms of connectivity, link sizing becomes more of an issue. The RESIZE subroutine speeds up the initial determination of link size by removing trunks until a threshold blocking factor called TINYBF is exceeded.

The RESIZE subroutine performs the following steps for each link in the network under study:

- . The current link blocking factor is obtained from the LSTATA array, the accumulated statistics array.
- . Iteratively remove a link and calculate the new blocking factor until the link blocking factor exceeds the value of TINYBF, or the mandatory number of trunks in the link is violated.

- . If any trunks are to be removed from the link, RESIZE decrements the number of trunks to be removed by one so that the link size equals the value just prior to the blocking factor threshold being exceeded.
- . If the links are still to be removed, then RESIZE removes the calculated number of links and writes messages to the log file that indicate the actions taken by RESIZE.

NOTE: The routine, RESIZE, cannot totally remove a trunk. The smallest link size allowed is one.

ISIZE, the subroutine that calls RESIZE, must check and satisfy the following criteria before calling RESIZE:

- . The design must be utilizing primary routing only.
- . The call may only occur on the first design iteration.
- . All link blocking factors are less than the threshold limit held in TINYBF.
- . Only terrestrial trunks are present in all links.

## ROUTE(ICON1,ICON2,IFIVER) - SUBROUTINE

**PURPOSE:** CONSTRUCTS A ROUTING TABLE FOR EACH PARCEL OF TRAFFIC. THE ROUTES MAY BE SELECTED ON THE BASIS OF MINIMUM COST, DISTANCE, OR THE NUMBER OF LINKS.

**INPUTS:** ICON1 - Primary route selection criterion (Integer).  
ICON2 - Secondary route selection criterion (Integer).  
WTCOST - Switch-to-switch cost matrix (Real).  
KDIST - Switch-to-switch distances matrix (Real).  
NPREF - Matrix of disfavored routing paths (Integer).

**OUTPUTS:** NROUTE - Routing table that gives both the next node along the traffic parcel's path and the pathlength of the best route via that node.  
IFIVER (pronounced "I Five R") - Allows for warning messages to be written in the log file upon returning to the calling module ISIZE if an indefinite primary route is found.

**CALLED BY:** ISIZE.

**CALLS TO:** PURSAT.

### LOCAL VARIABLES:

#### Integer:

BIDX(4) - Indices of the best four routes between the switch pair under examination.  
I - Origination node.  
IDX, INDEX, INDEXK, INDEX1 - DO loop/array indices.  
J - Destination node.  
K, L, M - First, second, and third intermediate nodes, respectively.  
KVAL - Counter for the number of the first intermediate nodes.  
PTEST - Counter for the nodes with only one allowable alternate route.

#### Logical:

QUIKRT - Set to .TRUE. when primary path routing is to be used during routing table construction.

#### Real:

ADDCOST - Aggregate cost weighting factor for non-preferred links and nodes.  
ADDIST - Aggregate distance weighting factor for non-preferred links and nodes.  
ADLEN - Aggregate tandem weighting factor for non-preferred links and nodes.  
RPARAM(4) - Route vector, where:  
RPARAM(1) = First node on the route.  
RPARAM(2) = Cost of using the route.  
RPARAM(3) = Pathlength of the route.  
RPARAM(4) = Distance between the link endpoints.

TABLE(200,4) - Contains the set of all possible end-to-end routes. For I=1 to 25:

TABLE(I,1) = First node on the route after the Ith intermediate node.

TABLE(I,2) = Cost of using the Ith intermediate node.

TABLE(I,2) = Pathlength to the Ith intermediate node.

TABLE(I,3) = Pathlength to the Ith intermediate node.

TABLE(I,4) = Kilometric distance to the Ith intermediate node.

#### MODULE LOGIC AND DESCRIPTION:

The routing table is generated by expanding the route trees from the source node I to the destination node J via intermediate nodes K, L, M, etc. for as many steps as necessary and that are allowed. The routes with path length greater than or equal to 5 are treated as equivalent, because the routing table contains only a prioritized list of nodes connected directly to the source node. Therefore, the nodes which are more than 4 links away from the source node have a negligible influence on the prioritization. If a primary route with more than 4 links is found, a warning message is issued to the log file.

There may be many alternate routes, but the total path length of any given route can not be longer than 4 links. Hence, only three intermediate nodes are allowed between each switch pair. For source node I and destination node J, the possible intermediate nodes may be defined as only as follows:

K - First intermediate node away from the source node I.

L - Second intermediate node away from the source node I and directly connected to node K.

M - Third intermediate node away from the source node I and directly connected to node L. Node M is directly connected to the destination node J.

There are no other possible intermediate nodes between I and K, K and L, L and M, and M and J. The example above is for a long pathlength. It is possible for I to be directly connected to J, or for there to be only one or two intermediate nodes.

An exception to this routing selection is when the "quick-routing" process is used. This process specifies that all nodes have only one alternate route. Typically such routing is done only during the initial design of a network with full or very high connectivity. In the case of high connectivity, each link from the source node is tested to find one intermediate node that connects to the destination node. If none are found, then one more intermediate node is tested as well. If there is still no route that can be established, then it is assumed that the internodal connectivity has diminished sufficiently to require the normal routing process.

The Subroutine ROUTE consists of two functions for constructing the network routing table. Both of these functions are contained in a loop that steps through all network switch pairs. The first function determines whether or not quick routing is possible. If so, then the more complex path generation process is bypassed. The path with either direct routing or one intermediate node is then found and the routing table entry is made directly. The second function of ROUTE performs the more complex path generation process.

For both of these functions, the four best routes available are determined and placed in the array TABLE. The best routes are chosen by including the weighting factors ICON1 and ICON2. These weighting factors may have the following values:

<u>ICON1/ICON2</u>	<u>VALUE</u>
2	Minimum Cost
3	Minimum Path Length
4	Minimum Distance

ICON1 is the primary weighting factor, and ICON2 is the secondary factor that is used to break ties.

Each time a route to the destination node is found, the RPARAM array is set with its first four values: the first intermediate node, the cost of the route, the path length of the route, and the distance. The RPARAM array is then compared with the appropriate entry in the matrix TABLE. The outcome of this comparison is determined by ICON1 and ICON2. If two routes are found to have the same value for ICON1, then ICON2 is used to determine the best route.

The constraints on routes currently implemented are that no path can have the same node in it more than once, and that no route with more than one pure satellite link is allowed. The logical function PURSAT is used to indicate pure satellite links. Furthermore, routes containing non-preferred nodes (as specified in the matrix NPREF) are disadvantaged by adding a cost, distance, or length to the route if a non-preferred node is encountered.

The parameter KVAL is used to record the number of links in a route. When the best routes have been determined, their locations in the array TABLE are stored in the best routes array BIDX. These four entries are copied into the routing table NROUTE. After this copy is made, Subroutine ROUTE checks to see if the path length of the primary route is 50 or more. Path length is recorded as 50 when an undetermined path length of 5 or more is encountered. If a primary route is found to have an indeterminate length, then the flag IFIVER is set and ROUTE loops to examine the next switch pair (I,J).

COMPUTATIONAL NOTE: Although the actual values of the node numbers and the pathlengths are, in reality, integers, they are stored as real numbers so that they are compatible with the storage requirements of the cost matrix and the mathematical computations which take place in Subroutine ROUTE.



## SCOUT(I,J,LNOW,ISAT) - SUBROUTINE

**PURPOSE:** TO FIND AN ACCEPTABLE, PREFERENCED ROUTE, BETWEEN ANY TWO NODES. NETWORK PROTOCOLS, PREVIOUS ROUTING HISTORY, AND CLASS OF TRAFFIC ARE ALSO TAKEN INTO ACCOUNT.

**INPUTS:** I - Origination node (Integer).  
J - Destination node (Integer).  
LNOW - Pointer to the PHIST indicating starting node in path history (Integer).  
ISAT - Traffic class at the starting node (Integer).

**OUTPUTS:** Updated PHIST array.

**CALLED BY:** LOAD.

**CALLS TO:** PURSAT.

### LOCAL VARIABLES:

Integer:

CALT - Alternate route currently under consideration.

IHOP - Pure satellite flag, where:

IHOP = 1 means current hop is a pure satellite hop.

IHOP = 0 means current hop is not a pure satellite hop.

II - Node currently under consideration.

INDEX - DO loop/array index.

JJ - Next node in the path after II.

NSAT - Number of satellite hops used so far.

LL - Pointer into the path history array.

STALT - Alternate route in use at the beginning of SCOUT's execution.

Logical:

LFLAG - Validation flag that indicates validity of the current step.

### MODULE LOGIC AND DESCRIPTION:

The SCOUT subroutine finds a route for a traffic parcel originating at node I and traveling to node J. The intermediate node numbers are stored in LNOW, which may be I itself. The route is determined one link at a time, and in accordance with the network's switch routing protocols.

The particular significance of the protocols is with regards to how blocked, or overflow, traffic parcels are handled. If all alternate paths out of a node are incapable of handling the traffic parcel, then the overflow traffic parcel may be rolled back to the preceding node II along the parcel's route. If node II has routing protocol 1, then the traffic can be offered to the next alternate link from it. If node II has routing protocol 2, then the traffic must be rolled back to either a node with a routing protocol 1, or the originating node I before any routing may be attempted. Hence, routing protocol 2 supports alternate routing with rollback traffic rerouted only at the originating node. Routing protocol 1, on the other hand, supports alternate routing from any node with this protocol.

The key to the successful execution of the subroutine SCOUT is the path history array, PHIST. When the subroutine is called, the array PHIST must contain the correct route "constructed"

so far for the parcel being routed. The parcel may be at any node on its route, since it can be an overflow parcel. SCOUT then completes the routing information, writing it into PHIST and using the array's last element to return status information.

When finding a route, the subroutine SCOUT ensures that no loops are made (i.e. if a node is already in PHIST, it is not used again), and then checks on the usage of satellite links. Additionally a maximum pathlength parameter, MAXLEN, is set up in the common data area COM2. By definition, PHIST cannot handle a path of more than 16 nodes, including origin and destination. There for the maximum number of links per path is 15. Schematically this is seen as 16 nodes {1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G} and 15 links {1-2, 2-3, 3-4, 4-5, 5-6, 6-7, 7-8, 8-9, 9-A, A-B, B-C, C-D, D-E, E-F, F-G}. The limit could be raised by increasing the size of the PHIST array, but care should be taken in doing this as there are many references to the last element of the PHIST array. This 17th element of the PHIST array contains all of the status information.

Before commencing on route determination, three checks are conducted on the validity of the arguments and the relevant entries in the path history array. Route determination is then achieved by using a loop, which repeatedly examines the current alternate of the current node until the destination is reached or no more forward routing is possible.

The next alternate out of the current node is obtained. If no more alternates are available, then the variables must be set up so the next alternate node out of the previous node is obtained. This is achieved by simply resetting various pointers. However, before doing that the previous node's routing protocol must be examined to see if the node supports re-routing. If it does not, then the traffic is rolled back to the node prior to it in the traffic path. If the current node is the starting node, the routine returns with no path determined.

If an alternate is obtained, then the next node in the path may be read directly from the routing table. The step is examined using the following validity checks:

- IS THIS NEXT NODE ACTIVE? This corresponds to the ACTIVE/INACTIVE BITS associated with the routing tables at each of the network's switches. If the node is inactive, repeat for the next alternate.
- DOES THE STEP EXIST? This check is actually no longer necessary as it corresponds directly to the number of alternates available out of the current node. However, if the step does not exist, then handling is forced as if no more alternate exist.
- WOULD USING THE STEP RESULT IN ANY OF THE FOLLOWING:
  - THE MAXIMUM NUMBER OF ALLOWED SATELLITE HOPS BEING EXCEEDED
  - THE SYSTEM ATTEMPTING TO USE A NON-EXISTENT LINK
  - THE SYSTEM ATTEMPTING TO USE A LOOP IN THE ROUTE

NOTE: In practice, Step 2 is only performed when a routing table has been input by the user and the physical links do not correspond. This could also result from the application of damage to such a set of input data.

If any of the above occur, the step is invalid and the loop repeats for the next alternate route choice.

If the step is determined to be valid, and the destination node has not been reached, then the path length must be checked. The path length is limited by a hard-coded value of 15 or a user-specified value less than or equal to 15. If this maximum path length would be violated by choosing that step, then the loop is repeated with the next alternate exiting the current node. Once all criteria are met and the step is taken, then the next node is saved in the path history array, and the new current node is stored in LL, and the loop is repeated. The loop terminates when the traffic parcel reaches the destination node.

The path history array, PHIST, returns values that depend on the specific outcome of SCOUT's routing determination. If an acceptable route forward to the destination is found, SCOUT returns with the route set into PHIST, and PHIST(17,1,ISAT) is set to the total number of nodes involved in the route. If a route from the starting node cannot be found in a forward direction, PHIST returns with all the same values passed from the calling routine, except that PHIST(17,1,ISAT) is set to zero. In the event of an error in the call, PHIST(17,1,ISAT) is set to -1.

When protocol 2 is in operation, special handling of the traffic has to take place in rolling blocked traffic back towards the originating node. It is, however, necessary to load traffic up to the blockage and then handle it accordingly, since overflow traffic at each tandeming node may not be blocked on alternate routing. This means that SCOUT must have the facility to record and return a path up to the blockage, rather than up to the destination. When all switches have protocol 1, SCOUT always returns the highest preference route if one exists. In this case, PHIST is returned with a route to the blocked node, but PHIST(17,1,ISAT) is set to 100+(number of nodes).

## SMALL (IPLUS) - SUBROUTINE

**PURPOSE:** TO IDENTIFY AND DELETE ALL LINKS WHICH ARE SMALLER IN SIZE THAN THE SPECIFIED MINIMUM SIZE VALUE AND FOR WHICH DELETION IS POSSIBLE. IF DELETION OF THE SMALL LINKS IS NOT PERMISSIBLE, THEN THE SIZE OF THEM IS SET EQUAL TO THE MINIMUM ALLOWABLE SIZE.

**INPUTS:** IPLUS - Status flag that indicates action taken (Integer).

**OUTPUTS:** NTRUNK - Adjusted trunk sizes (Integer array).

**CALLED BY:** ISIZE.

**CALLS TO:** None.

### LOCAL VARIABLES:

#### Integer:

INDEX - DO loop index.

M, N - DO loop/array indices.

MM, NN - Endpoints of a trunk group to be deleted.

NGROUP(25) - Array containing the number of trunk groups connected directly to each network switch.

TOTRNK - Total trunks in link.

#### Real:

HCOST - Current highest cost of link.

PCOST - Cost of link.

### MODULE LOGIC AND DESCRIPTION:

Subroutine SMALL searches the network for trunk groups smaller than the specified minimum size. It considers group size only.

If a group is found to be too small, then link constraints are examined to determine if any trunks can be deleted. A trunk group may be deleted if its connectivity is not mandatory, and if its deletion does not reduce the number of groups at the end nodes below the required minimum number. If these conditions are met, then the trunk group's cost is compared with the highest cost for all of the links deleted so far. If the present value is higher than any previous cost value, then it is stored as the highest value to date. When all links have been considered, the link with the highest cost value which can be deleted, is deleted. The subroutine then executes the same process to consider all the links again. The deletions must be done one at a time because of the effect on the number of groups per node, and thus the number of groups which can be deleted before the minimum value is reached.

As the subroutine examines each link in the network, if a group is found to be too small but can not be deleted, for any of the aforementioned reasons, then the trunk group is set equal to the minimum size. This is done by adding an appropriate number of LTYPE trunks to the group, unless LTYPE connectivity is prohibited in GROUP.

The subroutine returns to the calling routine with IPLUS set to zero if no changes were made, IPLUS set to -1 if a group was deleted, and IPLUS set to +1 if size increases were made.

NOTE: Although design constraints are considered before deleting a group, the maximum channels per node criterion is not compared before increasing trunk sizes. This is acceptable since after any change is made to the network during the design process, all design criteria checks are repeated. The assumption is made then that if any node has its maximum number of groups and each of those groups has a minimum size, then the maximum number of channels for that node is not exceeded. A subsequent pass of the maximum number of channels check reduces (if necessary) the channels in the groups with more than the minimum size.

## SMOOTH(AIJ,VIJ,A,NEQ) - SUBROUTINE

PURPOSE: TO CALCULATE THE UNIQUE VALUES OF THE EQUIVALENT MEAN RANDOM TRAFFIC (A), AND THE EQUIVALENT NUMBER OF TRUNKS (NEQ), SUCH THAT WHEN RANDOM TRAFFIC OF THE MEAN "A" IS OFFERED TO "NEQ" TRUNKS, THE SMOOTHER CARRIED TRAFFIC HAS MEAN VALUE AIJ AND VARIANCE VALUE VIJ.

INPUTS: AIJ - Smooth traffic mean (Real).  
VIJ - Smooth traffic variance (Real).

OUTPUTS: A - Equivalent mean random traffic (Real).  
NEQ - Equivalent number of trunks (Integer).

CALLED BY: BLOCK.

CALLS TO: ERLS, ERROR.

### LOCAL VARIABLES:

Real:

AA - Interpolated value for A (temporary value).  
AL - Lower limit to value of A (temporary value).  
AMV - Difference between the carried traffic variance and the carried traffic mean.  
AU - Upper limit to value of A (temporary value).  
EPS - Blocked traffic mean.  
ESA - Blocking factor based on non-integral link (temporary value).  
ESSA - Blocking factor based on non-integral link (temporary value).  
ESI - Blocking factor based on non-integral link (temporary value).  
F1A - Iterated value of A (temporary variable).  
F1L - Iterated value of A (temporary variable).  
F1U - Iterated value of A (temporary variable).  
S - Non-integral equivalent number of trunks.  
SA - Iterated value for S.  
SEARCH - Precision required for iterative convergence.  
SL - Lower limit for S.

### MODULE LOGIC AND DESCRIPTION:

The subroutine establishes an upper and lower limit on the offered Poisson traffic value A, and then iteratively calculates the limits and A until the difference between the upper and lower limits for A is less than 0.01% of the value of A.

NOTE: Since S represents a non-integral number of trunks, the blocking factors during each iteration cycle are determined from calls to ERLS, rather than ERLNGB.

Only the integral value of S is used in the calculation of the argument NEQ, because this integral value is sufficiently accurate for subsequent comparisons with the link to which traffic mean AIJ is offered.

The iterative calculations of the SMOOTH subroutine are based on the two following equations for the carried mean AIJ and carried variance VIJ:

$$AIJ = A * [1 - ESA]$$

$$VIJ = AIJ - A * [S - AIJ] * ESA$$

where S is the non-integral number of trunks in the equivalent link,  
A is the equivalent random offered traffic and  
ESA is the blocking factor when A Erlangs is offered to S trunks.

For any unique pair of values for AIJ and VIJ there is exactly one combination of A and S which coincides with the required limits. This module solves the above equations iteratively to establish both a lower and an upper limit on A and S. It then interpolates until the margin of error is acceptable.

STUB(I,J,AIJ1,VIJ1,AIJ2,VIJ2,L1,L2) - SUBROUTINE

PURPOSE: TO CALCULATE CARRIED AND OVERFLOW TRAFFIC WHEN TRAFFIC IS OFFERED TO A LINK, MAKING APPROPRIATE PROVISION FOR THE CLASS OF THE OFFERED TRAFFIC, AND RECORDING THE STATISTICS.

INPUTS: I - Originating node number (Integer).  
J - Destination node number (Integer).  
AIJ1 - Mean offered class 1 traffic (Real).  
AIJ2 - Mean offered class 2 traffic (Real).  
VIJ1 - Variance of class 1 offered traffic (Real).  
VIJ2 - Variance of class 2 offered traffic (Real).  
L1 - Node from which the link is loaded (Integer).  
L2 - Node to which the link is loaded (Integer).

OUTPUTS: OFLARR - Updated value of the carried and the overflow traffic (Real array).  
LSTAT - Updated link statistics accumulator array (Real array).  
NSTAT - Updated node statistics array (Real array).  
PPSTAT - Updated point-to-point statistics array (Real array).

CALLED BY: LOAD.

CALLS TO: None.

LOCAL VARIABLES:

Integer:

KPAC,MPAC - Updated parcel counters.  
SAT - Number of satellite trunks in the link L1-L2.  
TER - Number of terrestrial trunks in the link L1-L2.

Real:

AIJS - Mean traffic offered to satellite trunks.  
AIJT - Mean traffic offered to terrestrial trunks.  
CIJS - Mean traffic carried on satellite trunks.  
CIJT - Mean traffic carried on terrestrial trunks.  
CIJ1 - Mean class 1 carried traffic.  
CIJ2 - Mean class 2 carried traffic.  
CVIJS - Variance of traffic carried on satellite trunks.  
CVIJT - Variance of traffic carried on terrestrial trunks.  
CVIJ1 - Variance of class 1 carried traffic.  
CVIJ2 - Variance of class 2 carried traffic.  
EXITM1 - Mean of class 1 exiting traffic.  
EXITM2 - Mean of class 2 exiting traffic.  
EXITV1 - Variance of class 1 exiting traffic.  
EXITV2 - Variance of class 2 exiting traffic.  
VIJS - Variance of traffic offered to the satellite trunks.  
VIJT - Variance of traffic offered to the terrestrial trunks.



## MODULE LOGIC AND DESCRIPTION:

The first part of the subroutine's source code is concerned with the determination of the amount of traffic that is offered to each type of medium. For a pure satellite link, class 2 offered traffic may only be offered to the satellite. For a terrestrial link, both class 1 and class 2 traffic are offered to the terrestrial trunks. In the case of a dual-media link, the zero-order random approximation is used which splits class 2 offered traffic between terrestrial and satellite components in proportion to the number of trunks of each type available. Therefore, this approximation models random behavior. All class 1 traffic is offered to terrestrial trunks only. Class 2 traffic is not allowed to overflow from one medium to another because of the zero-order approximation. Hence, class 2 overflow traffic is rerouted over the same medium that the carried traffic component was originally routed.

When this allocation of offered traffic has been made, traffic carried through each medium is calculated. It should be remembered that in the array LSTAT, the blocking and the variance blocking factors are stored by medium type. The formula used to calculate the traffic carried by each medium is given as follows:

$$\text{CARRIED TRAFFIC ON MEDIUM OF LTYPE} = (\text{OFFERED TO MEDIUM OF LTYPE}) * (1 - \text{BF})$$

where BF is the LINK BLOCKING FACTOR OF MEDIUM OF LTYPE.

Subroutine STUB then calculates the exiting, carried traffic of each class. Traffic exiting the link as class 1 is comprised of class 2 traffic that has been carried via the satellite link plus the class 1 traffic that was carried via the terrestrial link. These two elements are summed for the mean and variance components, EXITM1 and EXITV1, respectively. The traffic exiting as class 2 is comprised only of class 2 offered traffic that has been carried via the terrestrial link. The mean and variance of this traffic component are calculated and stored in the parameters EXITM2 and EXITV2, respectively. When the exiting traffic has been calculated, it is then placed in the array OFLARR for the link destination node to await routing to the next node.

The subroutine also calculates the class proportion factor, CPF. CPF is defined to be the proportion of class 1 traffic exiting each node which originated as class 1 offered traffic. The formula for this parameter is as follows:

$$\text{CPF} = \frac{\text{CLASS 1 TRAFFIC AT ORIGINATION NODE OF LINK}}{\text{CLASS 1 TRAFFIC AT DESTINATION NODE OF LINK}}$$

Note that if there is no exiting class 1 traffic, then CPF is defined to be 1 as there is no change in carried traffic. The parameter CPF is also stored in the OFLARR array for the node from which traffic is loaded. This is because CPF varies as a function of the route taken; if the route changes so does the value of CPF.

The subroutine calculates the overflow traffic types and stores these in the overflow array OFLARR as relating to the source end of the link. The class 1 overflow is defined to be the product of the class 1 traffic and the class 1, or terrestrial, blocking factor. If the class 1 offered traffic has a zero value, then the overflow traffic for class 1 is not calculated because if no traffic was offered then no traffic can be blocked. NOTE: In the case of a pure satellite link, the overflow array component, OFLARR(L1,1,1) would already contain a valid value for the class 1 traffic at node L1 awaiting alternate routing.

The class 2 traffic overflow is defined by the following formula:

$$\begin{aligned} & (\text{Class 2 traffic offered to the terrestrial link}) * \text{Terrestrial Blocking Factor} \\ & + (\text{Class 2 traffic offered to the satellite link}) * \text{Satellite Blocking Factor} \end{aligned}$$

As can be seen from the source e code, this is equivalent to the trunk weight mean blocking factor. It should be noted here that the value obtained for the overflowing traffic variance is not completely accurate, since using a variance blocking factor for the link assumes that the offered variance equals the sum of the carried variance and the overflow variance. However, in justification of this method, the variance blocking factors derived in Subroutine BLOCK are based on a reasonably accurate prediction of the variance carried. The result is that the overflow variance is slightly smaller than it should be, which is to say that the overflow traffic is seen as smoother than it really is. This error is compensated by the calculation of the link blocking factors for the overflowing links for the next iteration and subsequent traffic calculated to be carried on the overflow link.

Subroutine STUB goes on to calculate the amount of offered traffic of each class which is carried by the network. This calculation is done by subtracting the value of the mean and variance of the overflow traffic from the mean and variance of the offered traffic for each class.

Link statistics are updated in this subroutine as well. The carried traffic statistics are recorded for the class of the offered traffic on the link. Traffic statistics are recorded in the LSTATA array by traffic class, and not by medium type.

For each complete traffic loading cycle over one link, traffic parcels are loaded forward one link and the forward counter, KPAC, is incremented by 1. The terminating traffic for both classes is accumulated. When a parcel reaches the destination node, the destination counter, MPAC, is incremented by 1. The parcel counters, KPAC and MPAC, are used to obtain a measure of the model's performance and have no significance in the network engineering aspects of the model.

## XHI (MHI, NHI, BFMEAN, TGOS, IFLG) - SUBROUTINE

**PURPOSE:** TO COMBINE THE EFFECTS OF SUBROUTINES HIGOS AND NEXTHI. GIVEN A NETWORK PAIR, IT EITHER REDUCES THE GOS OF THAT PAIR OR IF THAT IS NOT POSSIBLE, IT FINDS AND REDUCES THE GOS OF THE PAIR WITH THE HIGHEST GOS LYING BETWEEN THE ORIGINAL PAIR'S GOS AND A SPECIFIED LOWER LIMIT, TGOS.

**INPUTS:** MHI, NHI - Endpoints of the high GOS link (Integers).  
BFMEAN - Mean link blocking factor (Real).  
TGOS - Lower limit of GOS (Real).

**OUTPUTS:** IFLG - Return status flag (Integer).  
NTRNK - Adjusted trunk size array (Integer array).

**CALLED BY:** ISIZE.

**CALLS TO:** HIGOS, NEXTHI.

### LOCAL VARIABLES:

Integer:

IFL - Internal flag.

### MODULE LOGIC AND DESCRIPTION

XHI calls HIGOS with the original input pair as arguments. If a trunk adjustment is made by HIGOS, then the routine returns a non-zero value for IFLG. IFLG is set to -1 if a satellite link is created, a +1 if a trunk is added or deleted, and a zero if no change is made.

If HIGOS could not adjust the trunk sizes for the original input pair, NEXTHI is called to identify the pair with the next highest GOS which is also higher than the lower limit TGOS. If there is no such switch pair, then the routine returns with IFLG still set to zero.

If NEXTHI identifies another pair, then HIGOS is called again with updated values of MHI, and NHI. If this call to HIGOS produces no result, NEXTHI is called again and so on until either an adjustment is made, or no further switch pair can be identified for possible adjustment.

If no action can be taken at all, IFLG is returned set to zero. The routine is set up to simplify the logic in Subroutine ISIZE, by providing a more powerful way of adjusting the network.

# XLO (MLO,NLO,BFMEAN,TGOS,IFLG) - SUBROUTINE

PURPOSE: TO COMBINE THE EFFECTS OF THE SUBROUTINES LOGOS AND NEXTLO. GIVEN A SWITCH PAIR, IT EITHER INCREASES THE GOS OF THAT PAIR OR, IF THAT IS NOT POSSIBLE, FIND AND INCREASE THE GOS OF THE PAIR WITH THE LOWEST GOS LYING BETWEEN THE ORIGINAL PAIR'S GOS AND A SPECIFIED UPPER LIMIT, TGOS.

INPUTS: MLO, NLO - Endpoints of the low grade of service link (Integers).  
BFMEAN - Mean link blocking factor (Real).  
TGOS - Upper limit of GOS (Real).

OUTPUT: IFLG - Return status flag (Integer).  
NTRNK - Adjusted trunk size array (Integer array).

CALLED BY: ISIZE.

CALLS TO: LOGOS, NEXTLO.

## LOCAL VARIABLES:

Integer:

IFL - Internal flag.

## MODULE LOGIC AND OPERATION:

LOGOS is called with the original input pair as arguments. If a trunk adjustment is made, the routine returns a non-zero value for IFLG. IFLG is set to +1 if changes are made but not deletions, -1 if at least one trunk deletion is made, and 0 if no action could be taken.

If LOGOS could not adjust the trunk sizes for the original input pair, NEXTLO is called to identify the pair with the next lowest GOS which is also lower than the upper limit TGOS. If there is no such switch pair, then the routine returns with the IFLG still set to zero.

If NEXTLO identifies another pair, then LOGOS is called again with the updated values of MLO and NLO. If this call to LOGOS produces no result, NEXTLO is called again and so on until either an adjustment is made or no further switch pairs can be identified for possible adjustment.

If no action can be taken at all, IFLG is returned set to zero. The routine is set up to simplify the logic within the subroutine ISIZE.

### 4.3 OUTPUT MODULES

The output formatter consists of 37 modules, including the main program OUTFMT. A data flow diagram of this segment is included in Appendix A, and the detailed module descriptions are given below in alphabetical order.



## **BUFFER(TYPE, BUFF, DATALN) - SUBROUTINE**

**PURPOSE:** PROMPTS THE USER FOR A REPLY THEN RECEIVES AND VALIDATES USER'S REPLY.

**INPUTS:** TYPE - Prompt message pointer (Integer).  
BUFF - Buffer to hold user's reply (Character).  
DATALN - Length of user's reply in BUFF (Integer).

**OUTPUTS:** Prompt message to user's screen.

**CALLED BY:** REPLY, NOSECT, PROCOPT, ERROR.

**CALLS TO :** ERROR.

### **LOCAL VARIABLES:**

#### **Character:**

CH - Used as a pointer to BUFF.  
LINE - Input buffer.  
PROMPT(9) - Prompt messages.

#### **Integer:**

BUFLEN - Maximum length of input passed to buffer.  
I,J,K - Loop variables and pointers.  
ICH - Numerical value of each character in the buffer string.  
MSSNUM - Pointer to prompt message.

#### **Logical:**

END - Set to .TRUE. at the end of processing.  
FOUND - Set to .TRUE. when data is found in the input buffer.

#### **Parameter:**

MAXLEN - Maximum length of input buffer.

### **MODULE LOGIC AND DESCRIPTION:**

The local variables are initialized and the input buffer is cleared with spaces. If the message type is greater than or equal to 10, it is set equal to 2. The relevant prompt message is then displayed depending on TYPE. The following are the options provided to the user:

PROMPT(1) = 'ENTER OPTION, RANGE OF OPTIONS, A-ALL, E-EXIT'  
PROMPT(2) = 'ENTER ABOVE OPTION'  
PROMPT(3) = 'ENTER 1 OR RANGE OF ITERATION NUMBERS'  
PROMPT(4) = 'ENTER ITERATION NUMBER'  
PROMPT(5) = 'HIT <RET> TO CONTINUE'  
PROMPT(6) = 'HIT <RET> OR D TO DUMP'  
PROMPT(7) = 'HIT <RET>, D - DUMP, W - SET WINDOW, E - EXIT'  
PROMPT(8) = 'ENTER SOURCE NODE OR RANGE OF SOURCE NODES'  
PROMPT(9) = 'ENTER DESTINATION OR RANGE OF DESTINATION NODES'

The input buffer, LINE, is checked for the user's response. If LINE equals a blank space, then the user has entered <RETURN>. A check is then made on TYPE. If TYPE is less than or equal to 4, <RETURN> is an illegal response and ERROR is called. Otherwise, END is set equal to .TRUE..

Provided, LINE is not a <RETURN>, the length of the response is calculated. The code searches LINE for the first character which is not a space or an open parenthesis. This is deemed to be the start of the string and FOUND is set to .TRUE.. If the pointer I reaches MAXLEN before finding the start of the string, then ERROR is called. Once the start of the string is located, MAXLEN and loop counter J are used to find the end and is deemed to have found the end when the character is neither a space or a close bracket. DATALN is then calculated using I and J. If TYPE equals 5, there is no user response required except a <RETURN>, and BUFFER is exited. If TYPE is not equal to 5, each character in the user response string is converted to its upper case equivalent. The BUFFER subroutine is then exited. Provided input exists, it is then copied from LINE into BUFF.



## CHRINT(CHVAL) - FUNCTION

PURPOSE: CONVERTS A CHARACTER TO ITS EQUIVALENT INTEGER VALUE.

INPUTS: CHVAL - Character to be converted.

OUTPUTS: Equivalent integer value.

CALLED BY: PSNFIL, PRCOPT.

CALLS TO : None.

### LOCAL VARIABLES:

Character:

TABLE(10) - Look up table.

Integer:

CHRINT - Function name. Result is returned through this parameter.

I - Loop variable.

Parameter:

MAX - Maximum number of items in table.

### MODULE LOGIC AND DESCRIPTION:

CHRINT is initialized to 99. Each of the table values is compared with the character CHVAL. If found, CHRINT is set equal to the value at that table position. If not, CHRINT returns a value of 99.

## CLOSE - SUBROUTINE

PURPOSE: THIS ROUTINE CLOSES AND EITHER SAVES OR DELETES THE OUTPUT FILE, DEPENDING ON WHETHER THERE HAS BEEN OUTPUT TO IT.

INPUTS: None.

OUTPUTS: None.

CALLED BY: OUTFMT.

CALLS TO : None.

LOCAL VARIABLES: None.

### MODULE LOGIC AND DESCRIPTION:

OUTFLG is set to zero if no data has been written to the output file. If this is so, the file is deleted. Otherwise, it is closed normally.

## CLR\_SCREEN - SUBROUTINE

PURPOSE:     CLEARS THE TERMINAL SCREEN.

INPUTS:      None.

OUTPUTS:     None.

CALLED BY:   DSPGEN, DSPSWT, DSPTAB, DSPSTS, DSPNOD, DSPTRF, DSPDAM, DSPMTX,  
              MENU1, MENU2, MENU3, MENU4, MENU5, MENU6, MENU7, MENU8.

CALLS TO :   None.

LOCAL VARIABLES: None.

### MODULE LOGIC AND DESCRIPTION:

    This routine clears the terminal screen with a format statement that initiates a series of  
<RETURN>.

## DISPLY(OPT) - SUBROUTINE

PURPOSE: CONTROLS THE DISPLAYING OR DUMPING OF DATA.

INPUTS: OPT - User's response to first menu (Integer).

OUTPUTS: None.

CALLED BY: OUTFMT.

CALLS TO : MENU2, MENU3, MENU4, MENU5, MENU6, DMPALL, NOSECT, DSPFIL.

### LOCAL VARIABLES:

Integer:

SECT - User's response to section of data to display or dump.

Logical:

END - Set to .TRUE. when user requests to exit.

### MODULE LOGIC AND DESCRIPTION:

OPT determines whether the user requires to see the data or to dump it to a file. If the latter is chosen, MENU2 is called to ask the user if the whole file is to be dumped or only selected sections. DMPALL is then called, and END is set to .TRUE..

If selective processing is required, MENU3 is called to display three major data areas. MENU4, MENU5, or MENU6 are called for further selection within each area according to the user's selection from MENU3.

Provided the user has not requested to exit, DSPFIL is called to dump or display sections of data. If QTCM is being operated in the design mode, DSPFIL is called once for each design iteration. In the analysis mode, DSPFIL is only called once.

Note that for design mode, NOSECT is called to output a message notifying the user that in order to conserve disk space only the last iteration of Sections 1 and 2 are printed.

## DMPALL - SUBROUTINE

PURPOSE: DUMPS ALL OF THE INPUT FILE TO THE OUTPUT FILE, SECTION BY SECTION.

INPUTS: None.

OUTPUTS: None.

CALLED BY: DISPLY.

CALLS TO : DSPFIL.

### LOCAL VARIABLES:

Integer:

ITER - Iteration number.

NOSBST(3) - Number of subsections in each section.

SECT - Section number.

### MODULE LOGIC AND DESCRIPTION:

The routine loops through each section, setting LSBST equal to the number of subsections in the current section, and LITFP to the number of iterations (section 2 only). DSPFIL is then called to write the section of data to the formatted output file.

## DSPANL(SUBSCT, ITER) - SUBROUTINE

PURPOSE: DUMPS ANALYSIS ITERATION DATA TO FILE.

INPUTS: SUBSCT - Subsection number (Integer).  
ITER - Iteration number (Integer).

OUTPUTS: None.

CALLED BY: DSPFIL.

CALLS TO : MENU7, DSPMTX, DSPTAB, DSPSTS.

### LOCAL VARIABLES:

Character:

HDR(3) - Headers for all three areas.

Integer:

FIRST - First matrix to be output.

I,J - Loop counters.

LAST - Last matrix to be output.

Logical:

HDRFLG - Used to control output of header.

### MODULE LOGIC AND DESCRIPTION:

The module sets up the header information and then calls the relevant routine depending on SUBSCT. If SUBSCT equals 1, then a check is performed on where the output is to be directed. If the output is being sent to a file, FIRST is set to 1 and LAST is set to 3 so that all matrices are output. If output is to be displayed on the user screen, MENU7 is called to enable the user to choose which media's trunk size matrix is to be displayed. After the user's choices are read, LAST is set equal to FIRST. Then from within a loop, DSPMTX is called to output the trunk size matrices for the FIRST through LAST matrices. If SUBSCT equals 2, then DSPTAB is called to output routing table. Otherwise, SUBSCT is assumed to be equal to 3, and DSPSTS is called to output general statistics.

## DSPDAM - SUBROUTINE

PURPOSE:     DISPLAYS OR DUMPS THE DAMAGE AND NON-PREFERENTIAL ROUTES.

INPUTS:       None.

OUTPUTS:     Damage and non-preferential routes to screen or file.

CALLED BY:    DSPINP.

CALLS TO :    MENU8, CLR\_SCREEN, MENU7, PSNFIL, DSPMTX, REPLY.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy variable.

MEDIA(3) - Media headers - satellite, terrestrial or military.

MESS(4) - Headers for damage sections.

#### Integer:

DEAD(200) - Local storage for relevant data.

FIRST - Pointer to first matrix to be displayed.

LAST - Pointer to last matrix to be displayed.

LFN - Local file number.

NOARR(200,2) - User and switch matrices.

NOTRNK(200,200,3) - Trunk matrix.

NPREF(200,200,200) - Matrix of non-preferred routes.

PRCFLG - Process flag set negative when user requests to dump a screen to file.

SUBSCT - Holds user's response to MENU8 options.

#### Logical:

DAMTRK(4) - Set to .TRUE. if trunk is not damaged.

NPRUTE - Set to .TRUE. if non-preferred route is not damaged.

### MODULE LOGIC AND DESCRIPTION:

The routine initializes the local variables and then reads in the user, switch, and trunk data. If a particular trunk is not damaged, the corresponding element of the array DAMTRK is set to .TRUE.. If there are any damaged trunks in any of the three media trunk matrices, then the DAMTRK flag is set to .TRUE. for the overall matrix. The non-preferred routing data is then read in with the items of the array NPRUTE set to .TRUE., if the corresponding items of the matrix NPREF is not equal to 0.

The main loop is then entered if SUBSCT is less than or equal to 4. If output is directed to the user's screen, MENU 8 is called within this loop with the parameter SUBSCT to hold the user's option. The user can choose to output damaged users, switches, trunks, non-preferred routes, or to exit. PRCFLG is used as a control variable. If the user chooses to exit, PRCFLG is set to -1. If output is directed to a file, then SUBSCT is incremented in preparation to write the damage matrix. If PRCFLG is still greater than or equal to 0 (i.e. if the user chooses to display further damage information), the inner loop is entered. Within this loop the damage and non-preferred routes are

displayed or output. If no damage or non-preferred routes have been located, a message indicating this is output. OUT, the pointer to the output device, is reset to its original value on leaving the routine.



## DSPDSN(SUBSCT) - SUBROUTINE

PURPOSE: CALLS RELEVANT ROUTINE TO DUMP TO A FILE DIAGNOSIS ITERATION DATA.

INPUTS: SUBSCT - Output section number 1 through 4 (Integer).

OUTPUTS: None.

CALLED BY: DSPFIL.

CALLS TO : DSPNOD, DSPTAB, DSPTRF, INDATA, STATS.

### LOCAL VARIABLES:

Character:

HDR(4) - Header information for sections.

Logical:

TIME1 - Set to .FALSE. if it is not the first time through DSPDSN.

HDRFLG - Set to .FALSE. if section header is not required. (Check this !!!!!!!!)

### MODULE LOGIC AND DESCRIPTION:

TIME1 and HDRFLG are both initialized to .TRUE., and HDR is initialized as follows:

HDR(1) = 'SECTION 1 - NODE STATISTICS'  
HDR(2) = 'SECTION 2 - LINK STATISTICS'  
HDR(3) = 'SECTION 3 - POINT TO POINT STATISTICS'  
HDR(4) = 'SECTION 4 - TRAFFIC INFORMATION'

If output is directed to a file, then a title of 'DETAILED ANALYSIS DATA' is output and HDRFLG is set to .FALSE.. The first time through the routine DSPDSN, TIME1 is set to .FALSE. Also, the first time through, INDATA and STATS are called to read in and calculate detailed network statistics. The common variable HEADER is then set equal to HDR(SUBSCT) and the relevant display routines are called depending on SUBSCT. If SUBSCT equals 1, then DSPNOD is called to output the node statistics. If SUBSCT equals 2, DSPTAB with type 2 is called to output link statistics. DSPTAB is also called but with type 3 if SUBSCT equals 3 to output point-to-point statistics. Otherwise, DSPTRF is called to output traffic information.

## DSPFIL(SECT,ITER) - SUBROUTINE

PURPOSE: POSITIONS THE FILE POINTER FOR THE INPUT FILE TO THE AREA OF DATA SPECIFIED BY THE PARAMETER SECT AND CALLS THE RELEVANT DUMP/DISPLAY ROUTINE FOR EACH SUBSECTION WITHIN THE SECTION.

INPUTS: SECT - Section number (Integer).  
ITER - Iteration number (Integer).

OUTPUTS: None.

CALLED BY: DISPLY, DMPALL.

CALLS TO : PSNFIL, DSPINP, DSPANL, DSPDSN.

### LOCAL VARIABLES:

#### Integer:

I - Subsection counter.  
INTERNO - Parameter for PSNFIL, holds iteration number.  
STATUS - Indicates whether data area in input file is found.  
SUBSCT - Subsection number.

#### Logical:

END - Set to .TRUE. if data is not found or when all data in section has been displayed or dumped.

### MODULE LOGIC AND DESCRIPTION:

The first section of code initializes the variable I to point to the required subsections and END is set to .FALSE..

The next section determines the value of SUBSCT so that PSNFIL is called to the correct area of concern. If section two is involved, the subsection has to be found in the correct iteration.

Finally, if STATUS is returned from PSNFIL with a value of zero, then the data has been found and the relevant display/dump handling routine is called. The subsection counter is incremented, and the above procedure is repeated until all requested subsections have been processed or an error occurs. When all data in a section has been displayed/dumped or if STATUS is -1 (indicating section cannot be found), END is set to .TRUE. and the routine is terminated.

## DSPGEN - SUBROUTINE

PURPOSE:     DISPLAYS GENERAL INFORMATION TO EITHER A SCREEN OR DUMPS IT TO A FILE.

INPUTS:       None.

OUTPUTS:     General program variables to screen or file.

CALLED BY:   DSPINP.

CALLS TO :   CLR\_SCREEN, REPLY.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy variable.

HDR - Title for all sections.

MEDIA(3) - Media headers - satellite, PTT, terrestrial.

TXT(23) - Message text for data in section 1.

TYPE(3) - Message text for routing criteria - cost, length, or distance.

UNDRLN - Underline characters.

#### Integer:

CRITER(2) - Routing criteria values.

DSNCNT - Maximum number of possible design iterations.

DUMMY2 - Dummy variable.

ICON1 - First criteria for routing table - based on minimum.

ICON2 - Second criteria for routing table - based on minimum.

LFN - Local file number.

LTYPE - Determines preferred media type for which the program is allowed to adjust connectivity and link sizing - satellite, PTT, or military.

MAXLEN - Maximum path length between source - destination.

MAXHOP - Maximum number of satellite hops.

NODE - Number of nodes in the network.

NOITS - Number of analysis iterations.

PRCFLG - Process flag set negative when user requests to dump a screen to file.

#### Real:

DGOS - Tolerance limit in meeting design objective.

E1FRAC - Distribution factor.

E2FRAC - Distribution factor.

EGOS - Distribution constraint for individual link GOS.

EMIN - Trunk creation parameter.

LBF - Link blocking factor.

LVBF - Link variance blocking factor.

PARC - Minimum parcel size initialized at.

PERCPROP - Percentage of offered load.

PROP - Offered load.

RGOS - Design objective for arithmetic mean GOS.

STABLE - Minimum value for network stability.

TINY - Minimum value for size of parcel.  
TINYBF - Minimum blocking factor.  
WTGMIL - Weighting for military costs.  
WTGSAT - Weighting for satellite costs.

#### MODULE LOGIC AND DESCRIPTION:

This routine reads and outputs the general program variables. It then initializes LFN to OUT. LFN is the pointer to the output device which needs to be reset to its original value on leaving the routine. PRCFLG is used as a control variable. Provided its value is not negative, the data are output to the terminal. As data are displayed on the terminal a screen at a time, REPLY is called to inquire if the user wishes to dump the screen to a file. PRCFLG is then set accordingly. If output mode is set to disk on entry, PRCFLG is automatically set negative after data has been output. The same process is applied to the second and third general information screens.

## DSPINP(SUBSCT) - SUBROUTINE

**PURPOSE:** CALLS A SPECIFIED ROUTINE TO DISPLAY/DUMP A SUBSECTION OF DATA THAT EXISTS IN THE INPUT DATA.

**INPUTS:** SUBSCT - Subsection number (Integer).

**OUTPUTS:** None.

**CALLED BY:** DSPFIL.

**CALLS TO :** DSPGEN, DSPTAB, DSPSWT, DSPDAM, DSPESB, MENU7, DSPMTX.

### LOCAL VARIABLES:

#### Character:

HDR3(2) - Headers for network cost subsections.

#### Integer:

FIRST - First matrix to be output.

I,J - Loop counters.

LAST - Last matrix to be output.

#### Logical:

HDRFLG - Used to control output of header.

MEDIA - Set to .TRUE. if subsections 5 or 6 are required.

### MODULE LOGIC AND DESCRIPTION:

If SUBSCT equals 1, DSPGEN is called to output general information. If SUBSCT equals 3, DSPTAB is called to output nodal switch information. If the subsection required is 5 or 6 (mandatory connectivity or trunk size), the data consists of several matrices, one for each media, therefore MEDIA is set to .TRUE.. The header information is initialized, and the relevant routine is called to display/dump the data. If subsection is 8, DSPSWT is called to output the routing table. If SUBSCT equals 9, then DSPDAM is called to output damage and preferred routing. DSPESB is called if subsection is 10 to output restrictions on satellite routing (Section 10 is no longer used. This code should be removed in the future). Subsections 2, 3, 4, and 7 do not require special processing.

If a matrix is to be output, DSPMTX is called. This is called within a loop so that if necessary, depending on media selected, one or more matrices can be output. Within the loop, the header is amended to include extra information. If required, MENU7 is called to ask the user which media are to be processed.

## DSPLNK(I, J, COUNT) - SUBROUTINE

PURPOSE:     DISPLAYS NETWORK LINK STATISTICS.

INPUTS:       I, J - Link endpoint counters (Integers).  
              COUNT - Number of network links with a non-zero number of trunks (Integer).

OUTPUTS:     Link statistics to the output device.

CALLED BY:   DSPTAB.

CALLS TO :   None.

### LOCAL VARIABLES:

#### Character:

TEXT(3) - Link statistics message text.

#### Logical:

TIME1 - Set to .TRUE. at first pair of link endpoints in network matrix. (This variable is never used and should be omitted in the future.)

#### Real:

BF(2) - Mean link blocking factor.  
GM(2) - Gross mean offered traffic for specified link.  
GV(2) - Variance of gross offered traffic for specified link.  
LGOS - Average link grade of service for terrestrial traffic.  
NM(2) - Mean of net carried traffic over specific link.  
NOT - Intermediate variable in the calculation of LGOS.  
NV(2) - Variance of net carried traffic over specific link.  
OCC - Link occupancy.  
RM(2) - Mean rollback traffic for link.  
RV(2) - Variance of rollback traffic for link.

### MODULE LOGIC AND DESCRIPTION:

The TEXT array is initialized, and the statement function ADD(I,J,K,L) is defined to be equal to LSTATATA(I,J,K,L) plus LSTATATA(I,J,K,L). If I equals 1 and J equals 2, then TIME1 is set to .TRUE..

The main loop of DSPLNK is then initiated if the number of terrestrial trunks in the specified link is not 0. COUNT is then incremented by 1. Mean and variance of gross, rollback, and net link traffic is then calculated for the satellite and terrestrial portions of the specified link. These calculations utilize the defined ADD statement function. NM and NV are calculated by subtracting the rollback traffic mean and variance from the gross carried traffic mean and variance, respectively. BF, or link blocking factor, is set equal to LSTATATA, where L = 5.

The mean link occupancy, OCC, is determined by dividing the sum of the net terrestrial and satellite traffic by the number of trunks per link. The link grade of service LGOS is then calculated.

If the mean link blocking factor of terrestrial traffic is 1, then LGOS is set equal to 1. Otherwise, LGOS is determined using the following algorithm:

$$LGOS = \frac{NOT - (Gross\ Carried\ Traffic - Rollback\ Traffic)}{NOT}$$

$$\text{where } NOT = Gross\ Offered\ Traffic - \frac{Rollback\ Traffic}{1 - Blocking\ Factor}$$

After the LGOS is determined, the link statistics are written to the output device, and DSPLNK is exited.

## DSPMTX(ITER, SECT, SUBSCT, APTR) - SUBROUTINE

PURPOSE:     DISPLAYS ALL OR PART OF A MATRIX TO THE SCREEN OR DUMPS ALL OR PART OF IT TO THE OUTPUT FILE.

INPUTS:     ITER - Iteration number (Integer).  
             SECT - Section number (Integer).  
             SUBSCT - Subsection number (Integer).  
             APTR - Pointer to a particular area of matrix (Integer).

OUTPUTS:     None.

CALLED BY:   DSPINP, DSPANL, DSPDAM.

CALLS TO :   REPLY, CLR\_SCREEN.

### LOCAL VARIABLES:

#### Character:

CDESC - Message string.  
DUMMY1 - Dummy variable.  
OPTION - Currently not used, and should be omitted at a future date.  
UNDRNLN(150) - Underline for HEADER.

#### Integer:

I, J, K - Loop counters and pointers.  
IARR(200,200,5) - Set equal to LARR.  
IVAL - Integer value used to aid process control throughout DSPMTX.  
K1, K2 - Subsection counters.  
LFN - Local storage for output channel pointer.  
LSTITR - Last iteration number.  
LSTSBT - Last subsection read.  
LSTSCT - Last section read.  
NOARR(9) - Number of arrays in each subsection of input data.  
PRCFLG - Process flag.

#### Real:

RARR(200,200,5) - Set equal to LARR.

#### Logical:

FLAG - Data description for array type - integer (I) or real (R).  
LARR(200,200,5) - Matrix data to be displayed.  
LVAL - Logical value used in different ways throughout DSPMTX.

### MODULE LOGIC AND DESCRIPTION:

After initialization, the data are read into LARR. The model outputs sections of data by looping through one WRITE statement. A section can contain more than one matrix, therefore when reading in data the number of arrays read in must match those output. K1 and K2 are used to track the number of matrices per section. If the output device is the user screen, there is a restriction on the amount of data that can be displayed, therefore the window size is initialized. R1, R2, C1, and



C2 contain the limits for matrix display size. FLAG is used to indicate if the elements of the matrices are of type real ('R') or integer ('I').

If PRCFLG is greater than or equal to 0, the output loop is entered. The whole matrix (if sent to output file) or part of a matrix (if output to user screen) is output after the header information. REPLY is called so that the user can select another window, dump the screen to a file, or proceed. When proceeding, if part of a matrix has been output, the rest of the matrix is then displayed.

## DSPNOD - SUBROUTINE

PURPOSE: DUMPS OR DISPLAYS THE NODE STATISTICS.

INPUTS: None.

OUTPUTS: Outputs node statistics.

CALLED BY: DSPDSN.

CALLS TO : CLR\_SCREEN, REPLY.

### LOCAL VARIABLES:

#### Character:

TEXT(3) - Headers for screens.

DUMMY1 - Dummy variable.

UNDRLN(65) - Underlines.

#### Integer:

J, K - Loop counters and pointers.

LFN - Local file number.

PRCFLG - Process flag.

R1, R2 - Row limits.

SCREEN - Specifies either the originating/terminating traffic for node screen or the tandeming traffic for node screen.

#### Logical:

LEIO - Set to .TRUE. if all the data has been output.

LOPT - Indicates that user selected windowing or dumping and therefore loop must be repeated with existing values.

### MODULE LOGIC AND DESCRIPTION:

The local variables LNF, SCREEN, and PRCFLG are initialized to OUT, 1, and 0, respectively. The outer loop of the routine is entered if SCREEN is less than or equal to 2 and PRCFLG is greater than or equal to 0. If the output is sent to a file, R2 is set to NNOD (the number of nodes). Otherwise, if the output is to be displayed to the user's screen, R2 becomes the row limit to avoid overflow of the screen buffer.

The inner loop is then entered provided the flag LEIO is set to .FALSE.. The header for the first screen 'ORIGINATING/TERMINATING TRAFFIC FOR NODE' is output along with the corresponding node statistics. REPLY is then called so that the user can select another window, dump the screen to a file or proceed to the next screen. The PRCFLG flag is used as a control variable. If PRCFLG equals 1, then processing continues and the rest of the matrix is displayed. Otherwise, if PRCFLG equals 0 (should be less than or equal to 0), the flag LEIO is set to .FALSE. and processing continues.

OUT, the pointer to the output device, is reset to its original value and the variable SCREEN is incremented so that the process is repeated for the 'TANDEMING TRAFFIC FOR NODE' screen.

## DSPSTS - SUBROUTINE

PURPOSE:     DISPLAYS OR DUMPS THE ANALYSIS ITERATION STATISTICS.

INPUTS:       None.

OUTPUTS:     Analysis iteration statistics to screen or file.

CALLED BY:   DSPANL.

CALLS TO :   CLR\_SCREEN, REPLY.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy variable.

#### Integer:

KPAC - Number of link loadings completed.

LFN - Local file number.

LPAC - Number of link roll backs completed.

MHI - Endpoint of range for highest point-to-point GOS.

MLO - Endpoint of range for lowest point-to-point GOS.

MPAC - Number of link loadings which reached destination.

NHI - Endpoint of range for highest point-to-point GOS.

NLO - Endpoint of range for lowest point-to-point GOS.

PRCFLG - Process flag set negative when user requests to dump a screen to file.

#### Real:

APLOST - Specially handled traffic.

BLOTM - Network blocked traffic.

CARSUM - Carried sum of squares.

CHNGE1 - Change for carried sum of squares.

CHNGE2 - Change for network blocked traffic.

CURITR - Summary iteration.

DMEAN1 - Delta mean for link blocking.

DMEAN2 - Delta mean for point-to-point GOS.

EGOS1 - Percentage point-to-point GOS within plus/minus EGOS.

EGOS2 - Percentage point-to-point GOS within plus/minus EGOS\*2.

LMEAN - Mean link blocking.

LVAR - Variance for link blocking.

PARC - Minimum parcel size.

PHI - Highest point-to-point GOS between MHI and NHI.

PLO - Lowest point-to-point GOS between MLO and NLO.

PMEAN - Mean point-to-point GOS.

PVAR - Variance for point-to-point GOS.

#### MODULE LOGIC AND DESCRIPTION:

The routine first initializes the local variables and then reads in the data. PRCFLG is used as a control variable. Provided the value of PRCFLG is not negative, the data are then output. If necessary, REPLY is called if the data are to also be dumped to a file. PRCFLG is set accordingly. OUT, the pointer to the output device, is reset to its original value on leaving the routine.

## DSPSWT - SUBROUTINE

PURPOSE:     DISPLAYS OR DUMPS THE NODAL SWITCH INFORMATION.

INPUTS:       None.

OUTPUTS:     Switch information to screen or file.

CALLED BY:    DSPINP.

CALLS TO :    CLR\_SCREEN, REPLY.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy variable.

HDR(5) - Title for all sections.

UNDRLN - Underlines characters.

#### Integer:

IMAT(200,5) - Data array.

LEN - Length of underline to be used.

LFN - Local file number.

PRCFLG - Process flag set negative when user requests to dump a screen to file.

### MODULE LOGIC AND DESCRIPTION:

After initialization of the local variables, node switch data are read. PRCFLG is used as a control variable. Provided the value of PRCFLG is not negative, the header and node numbers which are output and underlined. The node data are then output, and if being displayed, REPLY is called if the data are to also be dumped to a file. PRCFLG is set accordingly. OUT, the pointer to the output device, is reset to its original value on leaving the routine.

## DSPTAB(TYPE) - SUBROUTINE

**PURPOSE:** DISPLAYS ROUTING TABLE, LINK STATISTICS, OR POINT-PAIR STATISTICS TO THE OUTPUT DEVICE.

**INPUTS:** TYPE - Indicates type of table to be displayed (Integer).

**OUTPUTS:** Specified table to output device.

**CALLED BY:** DSPINP, DSPANL, DSPDSN.

**CALLS TO :** CLR\_SCREEN, DSPLNK, REPLY.

### LOCAL VARIABLES:

#### Character:

DUMMY1 - Dummy variable 10 characters long.

#### Integer:

COUNT - Running counter of number of rows displayed.

E1 - Least maximum number of rows in display and the number of network nodes.

IFLAG - Flag indicating whether the routing table has been updated or not.

INDEX - Length of routing table header.

LFN - Output device number.

NROUTE(200,200,8,2) - Routing table with alternate routes for terrestrial and satellite links.

PRCFLG - Process flag, which is set to -1 to exit the routine.

#### Logical:

DATOUT - Set to .TRUE. when data have been output.

END - Set to .TRUE. to end the DSPTAB routine.

TIMEi - Set to .FALSE. after any data are output.

### MODULE LOGIC AND DESCRIPTION:

After initialization, the value of TYPE is checked. If TYPE equals 1, then the routing table header is read into DUMMY1 and IFLAG. If IFLAG is 0, then the routing table is unchanged since the last update, or does not exist. The header is then searched for the integer '3'. If the header contains a '3', then no routing table exists. Likewise, if the header does not contain a '3', then the routing exists but has not been updated. An appropriate message is output, the user is prompted to enter a <RETURN> to continue, and PRCFLG is set equal to -1.

If IFLAG is not 0, then the updated routing table is read into the NROUTE matrix, and the DSPTAB routine continues. A loop, controlled by PRCFLG, is entered to output the specified data. The specified data is determined by the value of TYPE. If TYPE is 1 or 3, then the routing table or point-to-point statistics are output directly. If TYPE is 2, DSPLNK is called to output the network link statistics.

After the data are output, END is set to .TRUE., and the output loop is exited. The user can then choose to dump the displayed information to a file, to set window, or to exit. PRCFLG is then set according to the user's choice.

## DSPTRF - SUBROUTINE

PURPOSE:     DISPLAYS TRAFFIC INFORMATION - OUTPUT SECTION 4.

INPUTS:       None.

OUTPUTS:      Traffic statistics to user screen or dumps it to a file.

CALLED BY:    DSPDSN.

CALLS TO :    CLR\_SCREEN, REPLY.

### LOCAL VARIABLES:

#### Character:

TEXT(3) - Screen title.

#### Integer:

J, K, L - Loop counters and pointers.

LFN - Local file number.

PRCFLG - Process flag.

SCREEN - Number of screen - 1, 2 or 3.

#### Real:

RT(3) - [Is not currently used. Should be omitted in next S/W version.]

PERCNT - Percentage of traffic using primary path.

TOTPCT - Percentage of total carried traffic over primary path.

### MODULE LOGIC AND DESCRIPTION:

This routine displays the traffic information in screens. The local variables are initialized and the main loop is entered. The loop is repeated and the process flag, PRCFLG, is set to 0 at the beginning of each screen display process. The inner loop to display and/or dump data is entered if PRCFLG is greater than or equal to 0.

Within the inner loop, the header is output along with the title for the current screen. If SCREEN equals 1, the primary path data are totaled and output. If SCREEN equals 2, the erlang distributions are output. Otherwise, SCREEN equals 3 and the distributions for switch pairs are output. If output is directed to the user's screen, REPLY is called to receive the user's response to dump the screen to a file or to enter a return. If the user chooses to dump the screen, the information is output to a file and execution returns to the beginning of the inner loop. Otherwise, REPLY returns a value of -1 for PRCFLG, the output channel is reset to its original value, and DSPTRF is exited. If output is not directed to the screen, PRCFLG is set to -1, the output channel is reset to its original value and DSPTRF is exited.

ERROR(EPTR, EVAL, TYPE, \*) - SUBROUTINE

PURPOSE:     OUTPUTS AN ERROR MESSAGE.

INPUTS:     EPTR - Error message index number (Integer).  
             EVAL - Error value (Real).  
             TYPE - Type of error (Integer).  
             \* - Alternate return specifier (Integer).

OUTPUTS:     Error message to user's screen.

CALLED BY:   OPNFIL, PSNFIL, PRCOPT, REPLY, BUFFER.

CALLS TO :   BUFFER.

LOCAL VARIABLES:

Character:

DUMMY1 - Dummy parameter for BUFFER.  
ELINE - 60-character buffer that stores the error message text.  
ERRMSS(10) - Error message array (currently only 7 exist).  
TEXT(4) - Additional error information array.

Integer:

DUMMY2 - Dummy parameter for BUFFER.  
I - Counter used for length of error message, and as source node number.  
IFAC - Stores maximum node number limit.  
J - Destination node number, if applicable.  
K - Alternate route number, if applicable.  
LEN - Length of error message.

Logical:

END - Set to .TRUE. when error message decoding is finished.  
ENDSTR - Set to .TRUE. when the end of the error message is located.

Real:

VAL - Stores value of EVAL while error message type is determined.

MODULE LOGIC AND DESCRIPTION:

The variables ENDSTR and I are initialized to .FALSE. and 45, respectively. The length of the specific error message (up to 45 characters) is determined and stored in LEN. TYPE determines how to process the error value. If TYPE equals 0, then no error value exists and only the error message text is displayed. If TYPE equals 1, the error message refers to incorrect values for general parameters, such as GOS limit. The error value is passed to ERROR via the parameter EVAL. This number is printed to the user screen along with the error message text. If TYPE equals 2, the error was generated by an erroneous node value, link endpoints, or alternate route numbers. ERROR determines which of these three by using modulo arithmetic. The appropriate error message is then printed to the user screen along with the erroneous values. Lastly, if the error was not 'illegal input', the error message is cleared in preparation for the next error. This is done by calling BUFFER and passing it dummy variables.



## INDATA - SUBROUTINE

PURPOSE:    READS IN THE DETAILED ANALYSIS DATA.

INPUTS:     None.

OUTPUTS:    None.

CALLED BY:   DSPDSN.

CALLS TO :   None.

### LOCAL VARIABLES:

#### Character:

    DUMMY1 - Dummy variable containing 10 characters.

#### Integer:

    I, J, K, L - Loop counters and pointers.

    ITEMP(200,200) - Matrix used to temporarily store the military trunk statistics  
                     before they are combined with the terrestrial statistics.

#### Real:

    LSTAT(200,200,8,2) - Link statistics matrix.

### MODULE LOGIC AND DESCRIPTION:

    The purpose of this subroutine is to read the detailed analysis data. This is accomplished with a few looped read statements which read the node, link, point-to-point, and traffic statistics into COMMON data variables.

## MENU1(IANS) - SUBROUTINE

PURPOSE: FIRST MENU GIVES USER THE ABILITY TO DISPLAY OR DUMP DATA OR EXIT.

INPUTS: IANS - User's response (Integer).

OUTPUTS: First menu to user screen.

CALLED BY: OUTFMT (main program).

CALLS TO : CLR\_SCREEN, PROPT.

### LOCAL VARIABLES:

Character:

TEXT(3) - Menu choices displayed to user's screen.

Integer:

I - Loop counter.

Parameter:

MAX - Maximum number of options.

### MODULE LOGIC AND DESCRIPTION:

The screen is cleared and the title and menu are displayed. PROPT is called to process and validate the user's response. The first parameter in the call defines the prompt message. An example of this menu screen is shown below:

#### OUTPUT FILE HANDLER

\*\*\*\*\*

1. DUMP SECTIONS OF DATA TO A FILE
2. DISPLAY SECTIONS OF DATA
3. EXIT

## MENU2(IANS) - SUBROUTINE

PURPOSE:     DISPLAYS THE DUMP OPTIONS AND AN EXIT OPTION.

INPUTS:       IANS - User's response (Integer).

OUTPUTS:      Menu of dump options displayed to the user screen.

CALLED BY:    DISPLY.

CALLS TO :    CLR\_SCREEN, PROCOPT.

### LOCAL VARIABLES:

#### Character:

TEXT(3) - Menu choices displayed to user's screen.

TITLE - Menu title.

UNDLN - Underline.

#### Integer:

I - Loop counter.

#### Parameter:

MAX - Maximum number of options.

### MODULE LOGIC AND DESCRIPTION:

This routine calls CLR\_SCREEN to clear the user's screen and then displays the title and menu. PROCOPT is called to process and validate the user's response stored in IANS. The first parameter in the call defines the prompt message. An example of this menu screen is shown below:

#### DUMP OPTIONS .....

1. SELECTIVE DUMP
2. DUMP ALL THE FILE
3. EXIT

# MENU3(IANS) - SUBROUTINE

PURPOSE:     DISPLAYS THE THREE MAJOR AREAS IN THE OUTPUT FILE.

INPUTS:     IANS - User's response (Integer).

OUTPUTS:    Menu that displays major areas of the output file.

CALLED BY:   DISPLY.

CALLS TO :   CLR\_SCREEN, PRCOPT.

## LOCAL VARIABLES:

### Character:

TEXT(4) - Menu choices displayed to user's screen.

TITLE - Menu title.

UNDLN - Underline.

### Integer:

I - Loop counter.

### Parameter:

MAX - Maximum number of options.

## MODULE LOGIC AND DESCRIPTION:

CLR\_SCREEN is called to clear the user's screen and then the title and menu are displayed. PRCOPT is called to process and validate the user's response stored in IANS. The first parameter in the call defines the prompt message. An example of this menu screen is shown below:

### MAJOR AREAS OF OUTPUT FILE

\*\*\*\*\*

1. INPUT DATA
2. SUMMARY ITERATION DATA
3. DETAILED ANALYSIS DATA
4. EXIT

#### MENU4(OPT) - SUBROUTINE

PURPOSE:     DISPLAYS THE SUBSECTIONS WITHIN THE INPUT DATA AREA - SECTION 1.

INPUTS:       OPT - User's response (Integer).

OUTPUTS:      Menu that displays the subsections within the input data area.

CALLED BY:    DISPLY.

CALLS TO :    CLR\_SCREEN, PRCOPT.

#### LOCAL VARIABLES:

Character:

    TITLE - Menu title.

    UNDLN - Underline.

Integer:

    I - Loop counter.

#### MODULE LOGIC AND DESCRIPTION:

CLR\_SCREEN is called to clear the user's screen and then the title and menu are displayed. PRCOPT is called to process and validate the user's response stored in OPTS. The first parameter in the call defines the prompt message. An example of this menu screen is shown below:

#### DATA SECTIONS

\*\*\*\*\*

SECTION 1 - GENERAL INFORMATION

SECTION 2 - NETWORK COSTS & DISTANCES

SECTION 3 - ROUTING TABLE

SECTION 4 - MINIMUM TRUNK GROUP

SECTION 5 - MANDATORY TRUNK CONNECTIONS

SECTION 6 - TRUNK SIZE

SECTION 7 - OFFERED LOAD

SECTION 8 - SWITCH DETAILS

SECTION 9 - DAMAGE & PREFERRED ROUTING

SECTION 10- SATELLITE ROUTING CRITERIA (Commented out.)

## MENU5(OPT, MAXIT) - SUBROUTINE

PURPOSE:     DISPLAYS THE SUBSECTIONS WITHIN THE ANALYSIS DATA SECTION AND  
              ALLOWS USER TO SELECT 1 OR MORE OF THEM FOR OUTPUT.

INPUTS:       MAXIT - Maximum number of iterations (Integer).  
              OPT - User's response (Integer).

OUTPUTS:      Menu that displays the subsections within the analysis data section.

CALLED BY:    DISPLY.

CALLS TO :    CLR\_SCREEN, PRCOPT.

### LOCAL VARIABLES:

#### Character:

ITRMSS - Iteration message.  
TEXT(3) - Menu options.  
TITLE - Menu title.  
UNDLN - Underline.

#### Integer:

DUMMY - Dummy variable.  
I - Loop counter.

### MODULE LOGIC AND DESCRIPTION:

CLR\_SCREEN is called to clear the user's screen. The title and menu are then displayed. PRCOPT is called to process and validate the user's response which is passed back in the return argument. If there is more than one iteration of analysis data, the user is prompted for an iteration number. PRCOPT is called to obtain the iteration number. This is set up in a common block by PRCOPT and is not passed back. An example of this menu screen is shown below:

### SUMMARY ITERATION SECTIONS

.....

1. TRUNK SIZE DETAILS
2. ROUTING TABLE
3. SUMMARY STATISTICS

MENU6(OPT) - SUBROUTINE

PURPOSE: DISPLAYS THE DESIGN DATA SUBSECTION AREAS.

INPUTS: OPT - User's response (Integer).

OUTPUTS: Menu that displays the subsections within the design data section.

CALLED BY: DISPLY.

CALLS TO : CLR\_SCREEN, PRCOPT.

LOCAL VARIABLES:

Character:

TEXT(4) - Menu options.

TITLE - Menu title.

UNDLN - Underline.

Integer:

I - Loop counter.

MAX - Number of options.

MODULE LOGIC AND DESCRIPTION:

CLR\_SCREEN is called to clear the user's screen. The title and menu are then displayed. PRCOPT is called to process and validate the user's response which is stored in OPT. An example of this menu screen is shown below:

DETAILED ANALYSIS DATA  
\*\*\*\*\*

1. NODE STATISTICS
2. LINK STATISTICS
3. POINT TO POINT STATISTICS
4. TRAFFIC INFORMATION

MENU7(SUBSCT, IANS) - SUBROUTINE

PURPOSE:     DISPLAYS MEDIA OPTIONS.

INPUTS:       SUBSCT - Subsection which is calling this menu (Integer).  
              IANS - User's response (Integer).

OUTPUTS:     Menu that displays the media options - satellite, terrestrial, military, or overall.

CALLED BY:   DSPINP, DSPANL, DSPDAM.

CALLS TO :   CLR\_SCREEN, PRCOPT.

LOCAL VARIABLES:

Character:

    OPT - Fourth possible option or blank line.  
    TEXT(3) - Menu options.  
    TITLE - Menu title.  
    UNDLN - Underline.

Integer:

    I - Loop counter.  
    MAXOPT - Maximum number of options.

Parameter:

    MAX - Maximum number of options.

MODULE LOGIC AND DESCRIPTION:

The title is displayed after the screen has been cleared. The fourth option is then set up depending on the subsection calling the menu, and MAXOPT is set accordingly. PRCOPT is called to validate the user's response in IANS. An example of this menu screen is shown below:

OPTIONS WITHIN THIS SECTION

.....

1. SATELLITE
2. TERRESTRIAL
3. MILITARY
4. OVERALL



MENU8(OPTION) - SUBROUTINE

PURPOSE:     DISPLAYS AREAS OF NETWORK DAMAGE DATA.

INPUTS:     OPTION - User's response (Integer).

OUTPUTS:    Menu choices for damage and preferred routing.

CALLED BY:   DSPDAM.

CALLS TO :   CLR\_SCREEN, PRCOPT.

LOCAL VARIABLES:

Character:

TEXT(5) - Menu options.

TITLE - Menu title.

UNDLN - Underline.

Integer:

I - Loop counter.

Parameter:

MAX - Maximum number of options.

MODULE LOGIC AND DESCRIPTION:

This routine calls CLR\_SCREEN to clear the user's screen and then displays the title and menu. PRCOPT is called to validate the user's response stored in OPTION. An example of this menu screen is shown below:

DAMAGE AND PREFERRED ROUTING  
\*\*\*\*\*

1. USERS
2. SWITCHES
3. TRUNKS
4. NON-PREFERRED ROUTES
5. EXIT

#### NOSECT - SUBROUTINE

PURPOSE: PRODUCES A MESSAGE INDICATING THAT LACK OF DISK SPACE CAUSES SECTIONS 1 AND 2 TO BE AVAILABLE FOR ONLY THE LAST ITERATION OF DESIGN MODE.

INPUTS: None.

OUTPUTS: Message indicating data is not available for display due to lack of disk space.

CALLED BY: DISPLY

CALLS TO : BUFFER.

#### LOCAL VARIABLES:

Character:

CDUM - Used as a dummy argument to BUFFER.

Integer:

IDUM - Used as a dummy argument to BUFFER.

#### MODULE LOGIC AND DESCRIPTION:

Produces the message "DUE TO LACK OF DISK SPACE, SECTIONS 1 AND 2 ARE AVAILABLE FOR THE LAST ITERATION ONLY."

## OPNFIL(END) - SUBROUTINE

PURPOSE: OPENS INPUT AND OUTPUT FILES.

INPUTS: END - Set to .TRUE. if any errors occur during this routine [Logical].

OUTPUTS: None.

CALLED BY: OUTFMT.

CALLS TO : PSNFIL, DATE, TIME, ERROR.

### LOCAL VARIABLES:

#### Character:

CDATE - Date stamp.  
CTIME - Time stamp.  
DUMMY2 - Dummy variable for BUFFER.  
INFIL - Input file name.  
LINE - Used to read log file a line at a time.  
OUTFIL - Output file name.  
VERNUM - Version number.

#### Integer:

DUMMY2 - Dummy variable for BUFFER.  
I,J - Loop controller and counter.  
LLEN - Number of characters in LINE.  
LOG - Unit for log file.  
STATUS - Indicates success on finding data on input file.

#### Logical:

OUT - I/O channel for output file.

### MODULE LOGIC AND DESCRIPTION:

END is initialized to .TRUE. and is only reset if no errors occur. OUT, the location for output, is set equal to the output file. Both input files are opened and rewound. ERROR is called if neither of the files are found. The user is then prompted to enter an output file name. The output file is then opened. The program versions number VERNUM, the date stamp, and time stamp are written to the output file.

PSNFIL is called to find the first section of data in the input data file. STATUS is used to determine whether data is found and ERROR is called if not; otherwise, the number of nodes (NNOD) is read. This is necessary in order to read subsequent data. PSNFIL is then used to find the third area of data so that the number of design iterations can be read and displayed to the user. BUFFER is called so that the user can note the number and processing can then proceed.

The final area of code copies the log file to the output file line by line and the log file is then closed. END is reset to .FALSE. to indicate that everything has worked successfully.

## OUTFMT - PROGRAM

PURPOSE:     ALLOWS A USER TO DISPLAY OR DUMP ANY PART OF AN OUTPUT FILE  
              CREATED BY QTCM.

INPUTS:       None.

OUTPUTS:      Output files.

CALLED BY:    None. This is the main program for QTCM output formatter.

CALLS TO :    BLKDAT, OPNFIL, MENU1, DISPLY, CLOSE.

### LOCAL VARIABLES:

Integer:

    OPT - Holds user's response to menu.

Logical:

    END - Set to .TRUE. when user chooses to quit or when an error occurs while  
          opening initial files.

### MODULE LOGIC AND DESCRIPTION:

    This routine calls OPNFIL to open input and output files. MENU1 is called to display a menu which allows the user to dump sections of data to a file, to display sections of data to the screen, or to exit. If the user chooses to exit, END is set to .TRUE. and the routine is terminated. Otherwise, DISPLY is called to either display or dump data depending on user's response. Finally, CLOSE is called to close all files. If the files requested cannot be opened, END is set to .TRUE. and the loop is exited.

PRCOPT(TYPE, MAXVAL, REPLY) - SUBROUTINE

PURPOSE: PROCESSES USER REQUESTS.

INPUTS: TYPE - Used to indicate prompt message (Integer).  
MAXVAL - Maximum number of options (Integer).  
REPLY - User's response (Integer).

OUTPUTS: None.

CALLED BY: REPLY, MENU1, MENU2, MENU3, MENU4, MENU5, MENU6, MENU7, MENU8.

CALLS TO : BUFFER, ERROR.

LOCAL VARIABLES:

Character:

ALL - Set equal to 'ALL'.  
BUFF - User's response.  
EXIT - Set equal to 'EXIT'.

Integer:

FVAL - First node value.  
LEN - Number of characters in user's response.  
LVAL - Last node value.  
VAL - Accumulated numeric value processed from BUFF.

Logical:

END - Set to .TRUE. when BUFF has been processed.  
RANGE - Set to .TRUE. if a range of values is required.

MODULE LOGIC AND DESCRIPTION:

The outer loop, controlled by the logical variable END, is entered to process the user's response. The local variables are initialized and BUFFER is called to accept the user's reply. If TYPE is not 2 and the user's reply begins with an 'E' and is 4 letters or less, then the user's reply is compared to EXIT. If there is a match, END is set to .TRUE.; otherwise ERROR is called. If the user enters 'ALL', then FVAL, LVAL, and END are set to 1, MAXVAL, and .TRUE., respectively. ERROR is called if the user enters a word beginning with 'A' that is not 'ALL'.

If TYPE is equal to '2', the buffer is processed character by character. CHRINT is called to convert each character to an integer value. If the character is a number from 0 to 9, it is added to VAL. If a comma is encountered, it is assumed to be the first value of two. In this case, FVAL assumes the value of VAL. VAL is reset to zero and the flag RANGE is set to .TRUE.. If more processing is to be done, the second value is put into LVAL.

If TYPE is 2 or 4, only one value is required. If LVAL is not zero, ERROR is called, otherwise REPLY is set to FVAL. If LVAL is zero, it is set to FVAL as only one value is required. If either value exceeds the maximum permissible, MAXVAL, ERROR is again called. Lastly, the first and last section, iteration, and row and column limits are defined based on the value of TYPE.

PSNFIL(SECT, ITERNO, SUBSCT, STATUS) - SUBROUTINE

PURPOSE: POSITIONS THE FILE POINTER TO THE REQUESTED SECTION OR SUBSECTION OF DATA.

INPUTS: SECT - Section number (Integer).  
ITERNO - Iteration number (Integer).  
SUBSCT - Subsection number (Integer).  
STATUS - Indicates whether data is found (Integer).

OUTPUTS: File pointer set to the requested section or subsection of data.

CALLED BY: DSPFIL, OPNFIL, DSPDAM.

CALLS TO : ERROR, CHRINT.

LOCAL VARIABLES:

Character:

DUMMY1 - Dummy variable.  
ITERLN - Iteration header.  
LINE - Input file buffer.  
MJRHDR(3) - Major data section headers.  
SBSCLN - Subsection header.

Integer:

ANLITR - Analysis iteration number.  
CHRINT - Function name. Result is returned through this parameter.  
CURITR - Current iteration number.  
CURPSN - Current position of file.  
CURSCT - Current section of data.  
CURSUB - Current subsection.  
REQPSN - Required position.

Logical:

FOUND - Set to .TRUE. when subsection data is found.  
MJRSCT - Set to .TRUE. when section data is found.

MODULE LOGIC AND DESCRIPTION:

This routine initializes local variables and rewinds the input file if necessary. The file is then read until the major section header required is found. If the section is not encountered, ERROR is called. Otherwise, MJRSCT is set to .TRUE. and this loop is left. If SECT equals 2, then analysis data is sought and the correct iteration must be found. If the required iteration cannot be found, ERROR is called. Otherwise, FOUND is set to .TRUE. and this loop is left.

Finally, the subsection must be found, and again ERROR is called if the subsection cannot be found. Otherwise, CURSUB is calculated and compared to SUBSCT. If they are equal, FOUND is set to .TRUE. and this loop is left. If STATUS is negative, the major section does not exist. If the value is 1, the subsection cannot be found. If STATUS equals 0, then processing continues.

## REPLY(OPT, ACTION) - SUBROUTINE

PURPOSE: PROCESSES USER'S RESPONSE WHEN DISPLAYING DATA TO THE SCREEN.

INPUTS: OPT - Prompt message number and processing protocol (Integer).  
ACTION - Process flag (Integer).

OUTPUTS: None.

CALLED BY: DSPGEN, D3PCWT, DSPTAB, DSPSTS, DSPNOD, DSPTRF, DSPDAM, DSPMTX.

CALLS TO : BUFFER, ERROR, PRCOPT.

### LOCAL VARIABLES:

#### Character:

DUMP - Set equal to 'DUMP'.

EXIT - Set equal to 'EXIT'.

LINE - User buffer.

WINDOW - Set equal to 'WINDOW'.

#### Integer:

DUMMY - Dummy parameter for PRCOPT.

LEN - Length of user's response in LINE.

### MODULE LOGIC AND DESCRIPTION:

The process flag ACTION is initialized to 0 and BUFFER is called to accept the user's response. The response is stored in the buffer LINE and the number of characters in the response is stored in LEN. If LEN is returned from BUFFER as a 0, OPT is checked for equivalence to 6. If OPT equals 6, then only a <RETURN> is required and entered from the user to continue displaying information. ACTION is set to -1 and REPLY is exited. If LEN equals 0 and OPT is not 6, then ACTION is set equal to 1 and REPLY is exited. This causes the calling display routine to repeat the display of the previous screen. ACTION is set to 1 if dumping is required or -1 if exit is required. If the WINDOW option is chosen, PRCOPT is called for the range of rows to be displayed and again for the range of columns. OUTFLG is set to 9999 to indicate that data is being output to the file.

If LEN is not equal to 0, the user response consists of a character string. Checks are then conducted to determine if the user wishes to EXIT, WINDOW, or DUMP. If the OPT is equal to 7 and the user entered EXIT, then ACTION is set equal to -1 and REPLY is exited. If OPT is 7 and the user requested a WINDOW, then OUT is set to be the user screen and PRCOPT is called twice. These 2 calls obtain the source and destination node ranges for the window. If OPT is greater than 7 or equal to 6 and user requests to DUMP, the output is set to disk and OUTFLG is set to 9999. If OPT is less than 6, then ERROR is called to display the message of illegal input to the user screen.

## STATS - SUBROUTINE

PURPOSE: TO CALCULATE NODE STATISTICS AND TRAFFIC DISTRIBUTIONS.

INPUTS: None.

OUTPUTS: None.

CALLED BY: DSPDSN.

CALLS TO : None.

### LOCAL VARIABLES:

Real:

FLSHPT - Constraint value for peak point-to-point grade of service (GOS).

IMMPNT - Constraint value for immediate point-to-point GOS.

### MODULE LOGIC AND DESCRIPTION:

After all node statistic parameters are initialized, the node statistics from NSTAT are totaled and stored in TOTNST.

Another loop is then entered to calculate the remainder of the node statistics for each switch pair in the network. These node statistics include point-to-point GOS, link blocking factors, and total number of trunks for each media type.

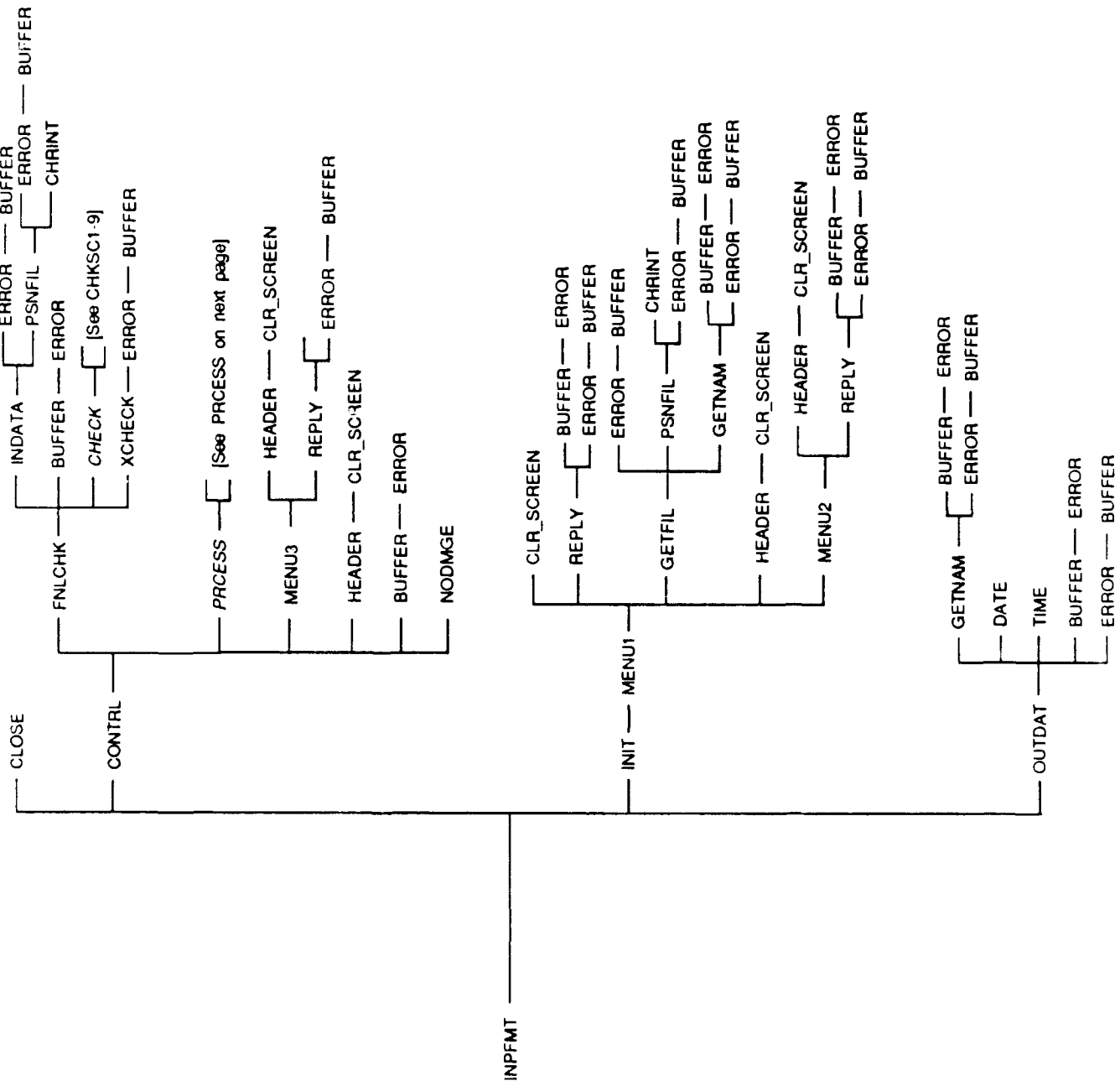
When the node calculations are completed, the total primary traffic and traffic distributions are computed. STAT passes the results of the node and traffic calculations to other output routines through the use of common blocks D1, D3, D4, and S1.



## APPENDIX A

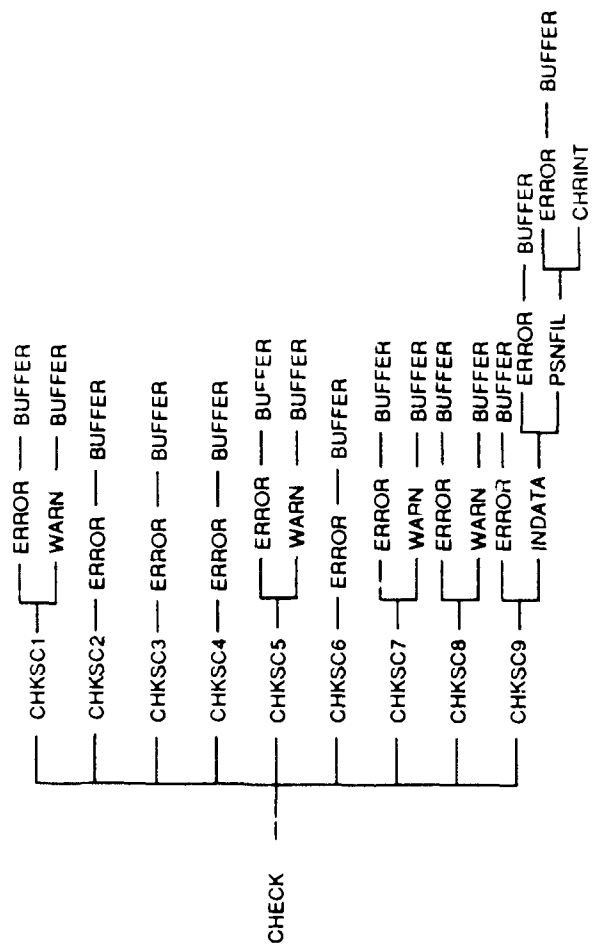
Data flow diagrams for each of the three QTCM segments are included in this appendix.

### INPUT FORMATTER FLOW DIAGRAMS -



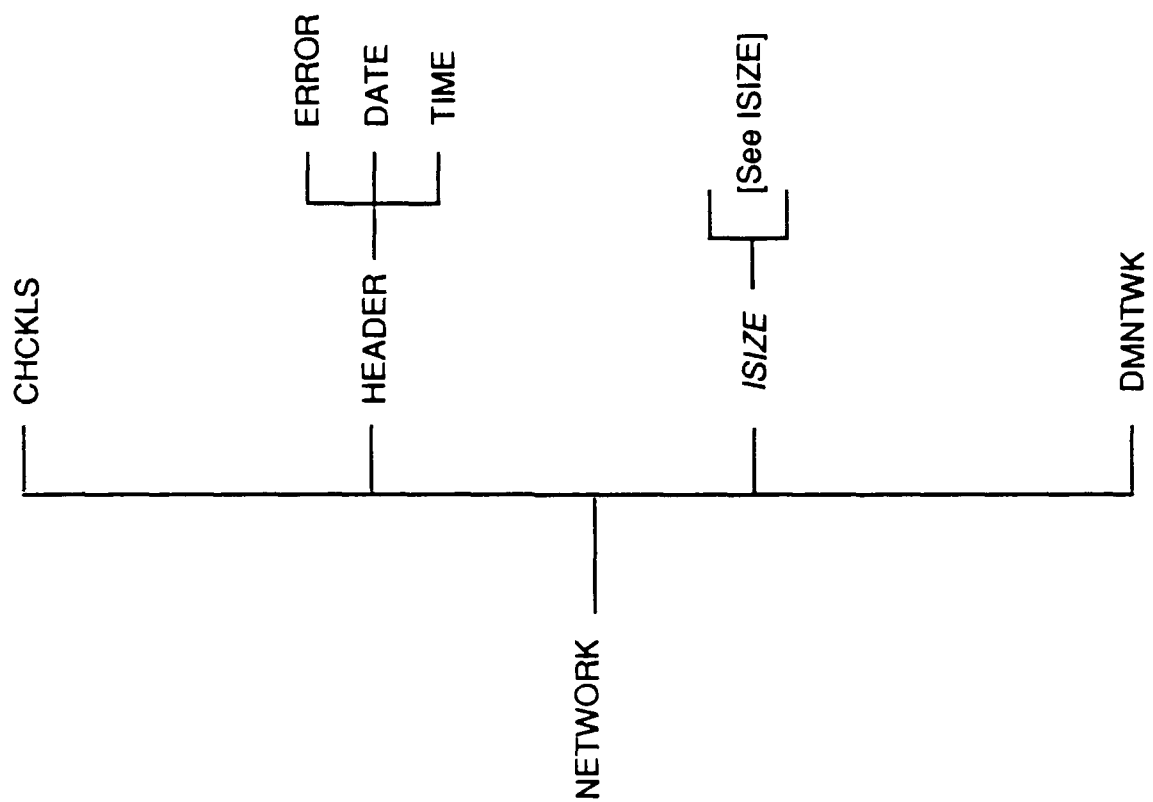


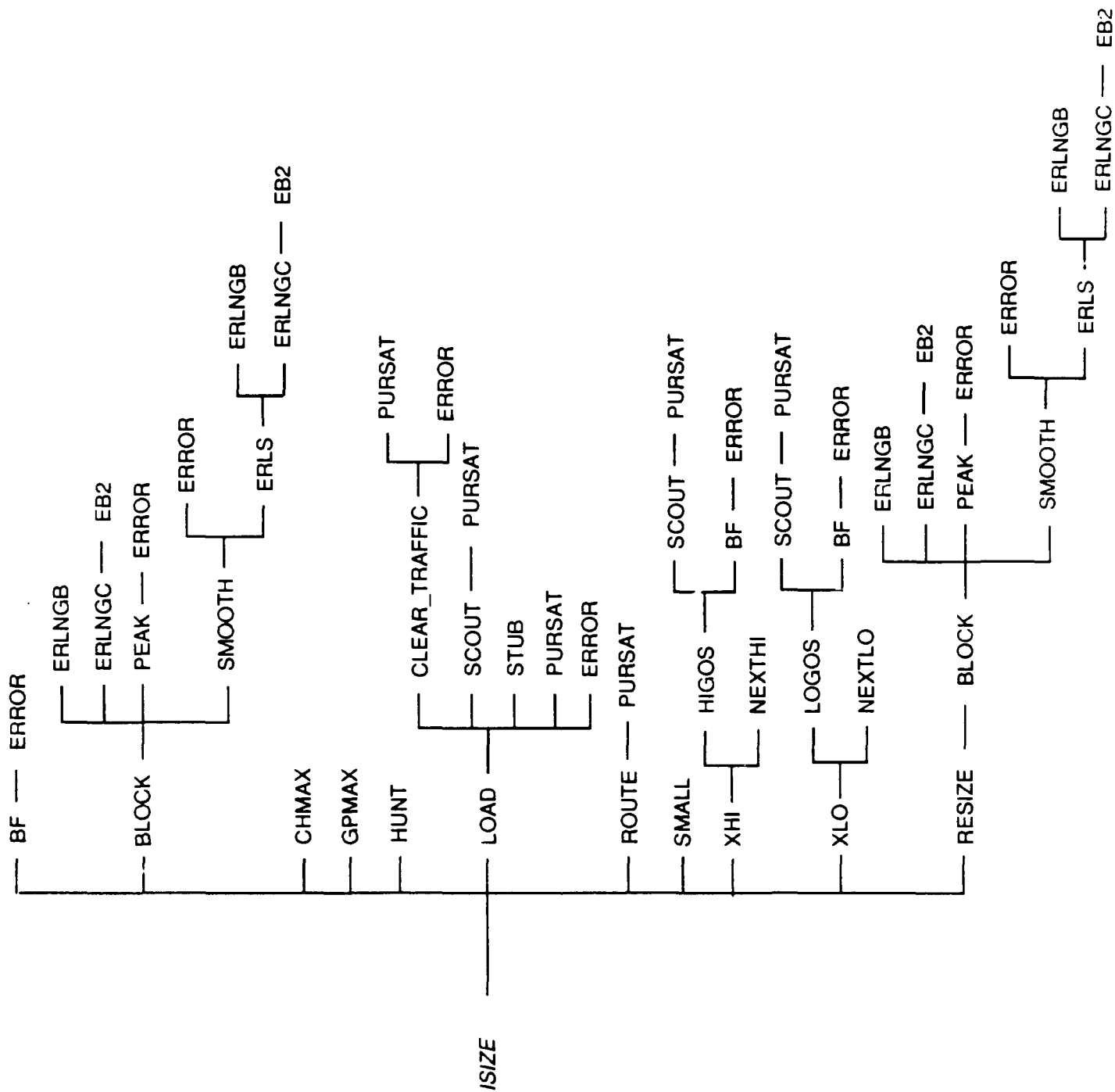




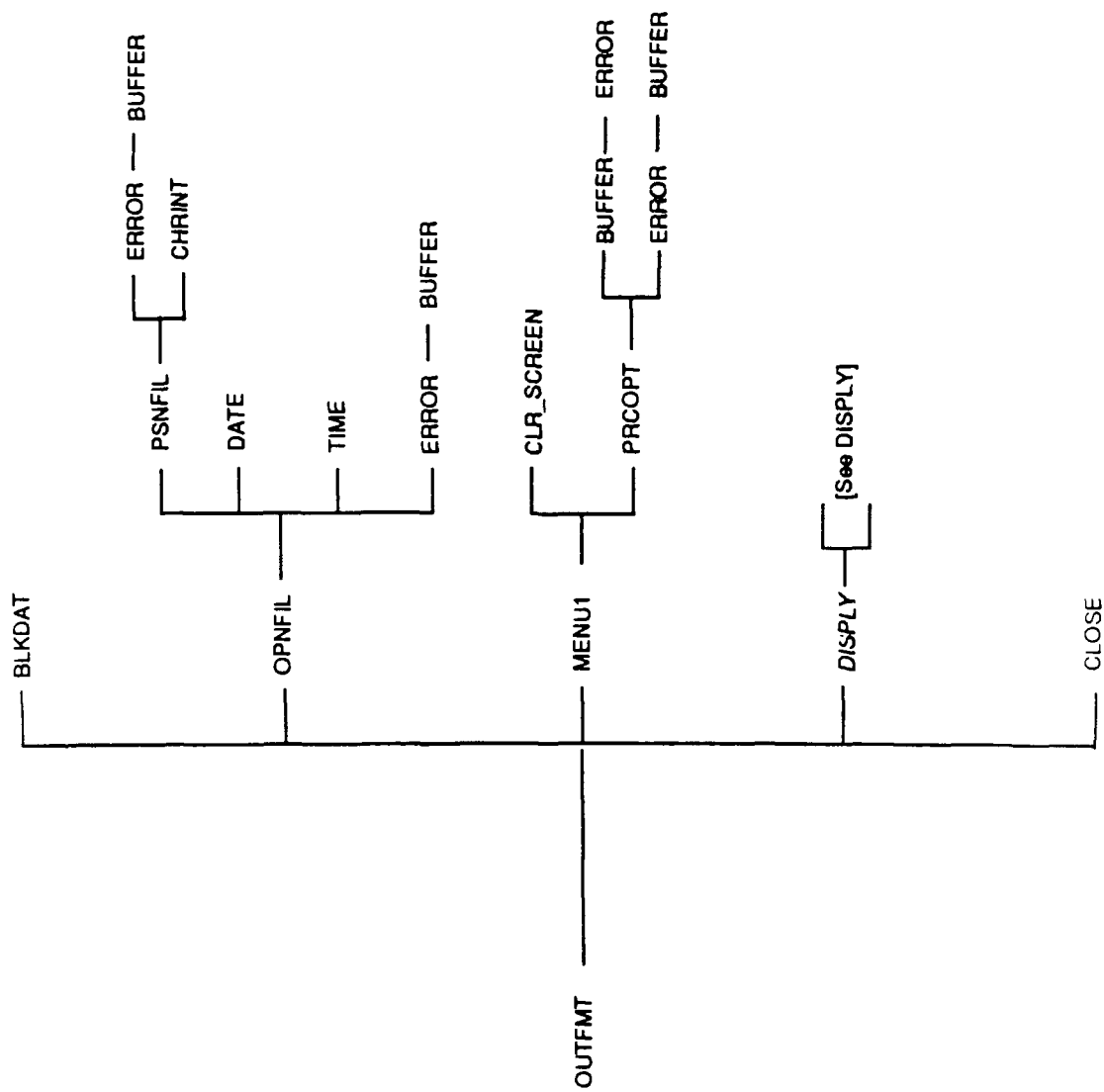


# NETWORK SIMULATION FLOW DIAGRAMS .



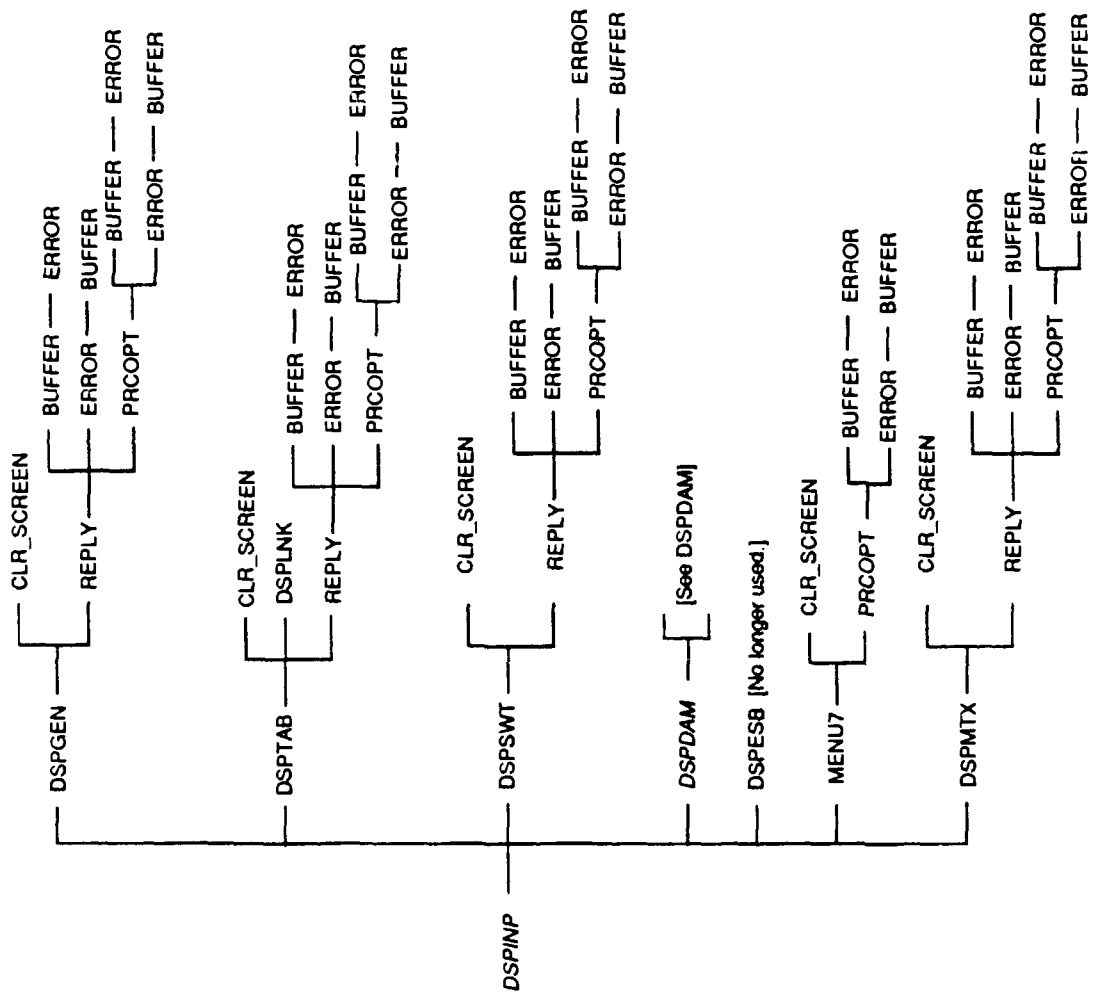


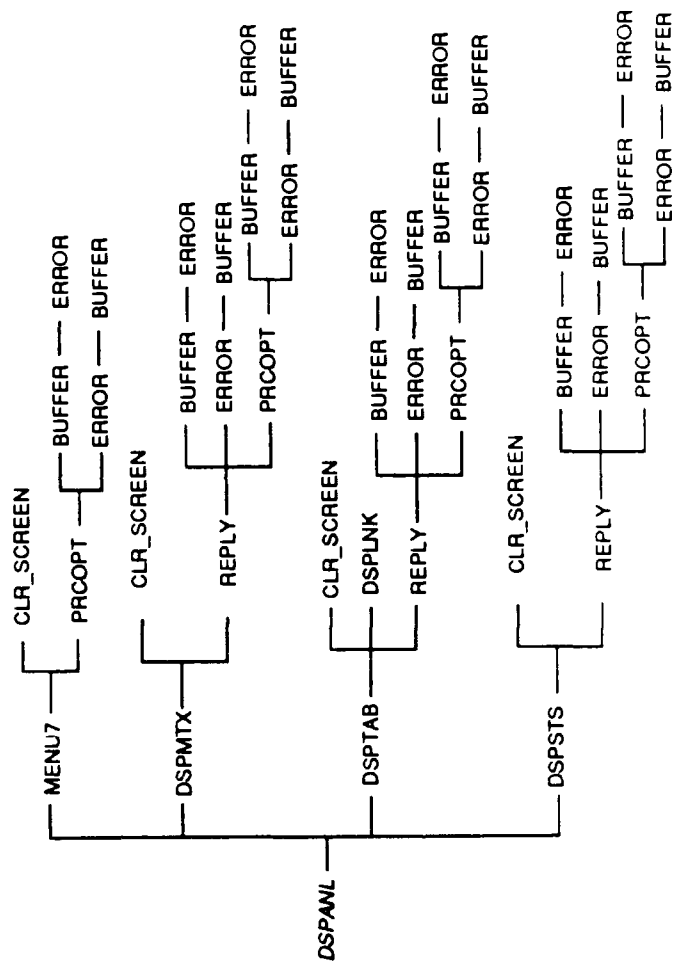
# OUTPUT FORMATTER FLOW DIAGRAMS -

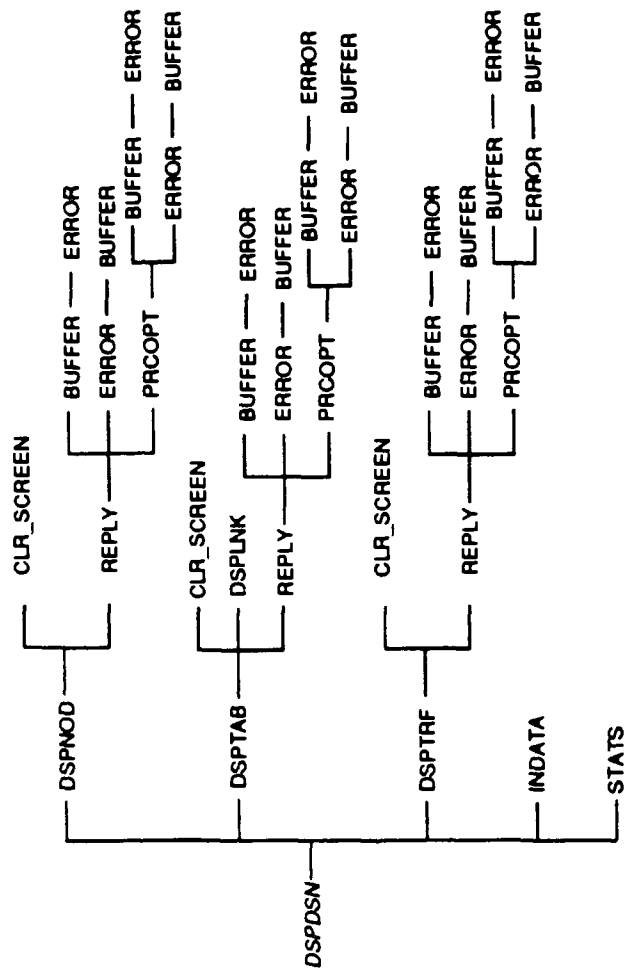


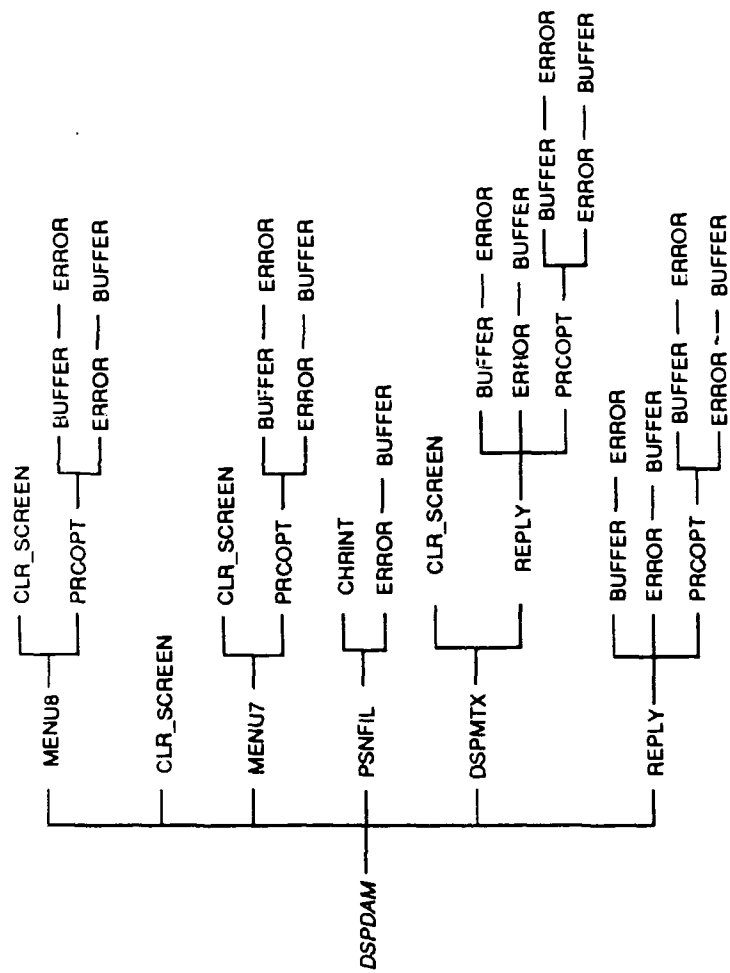












## APPENDIX B

Program NETWORK is the central part of a three-part analytical system. NETWORK simulates the network under study and generates traffic statistics. Listed below are common data variables used by NETWORK and a description of each:

### COMMON BLOCK BLANK

ROUTING TABLE ARRAY - NROUTE (I,J,K,L) [Integer]

I = Source node.

J = Destination node.

When L=1 K = Number ( $0 < K < 8$ ) of links from node I to J.

When L=2 K = Minimum path length, measured in tandemed links in the primary path route for each link identified at the start of the routing table.

Note: When there is no link, the path length is set to 99.

PATH HISTORY ARRAY - PHIST (I,J,K) [Integer]

When  $0 < I < 16$  I = Source node of each link in the path between switch pairs.

When J = 1 PHIST contains the value of the nodes sequentially encountered in the complete first choice route from the source node to the destination node and the sequence is reflected by their I address.

When J = 2 PHIST contains the number of the alternate path at the node address given by J=1, i.e. PHIST(I,1,K).

When I = 17 and

When J = 1 PHIST contains the number of valid entries of class K.

When J = 2 Not functional at present.

When K = 1 The path contains satellite links. Only used when toggled on and when for a specific I,J it has not already been toggled.

When K = 2 The path does not contain satellite links yet and the primary array is toggled on first.

Note: Dimension of 17 and storage of one unit means we can only have 16 switches and therefore 15 links between origin and destination.

TRUNK GROUP CONSTRAINTS - GROUP(I,J,K) [INTEGER]

I = Source switch.

J = Destination switch.

When K = 1 GROUP(I,J,1) is the minimum link size in number of trunks; signalling channel excluded.

When K = 2 GROUP(I,J,2) is the mandatory connectivity constraints for satellite trunks.

When K = 3 GROUP(I,J,3) is the mandatory connectivity constraints for PTT trunks.

When  $K = 4$   $GROUP(I,J,4)$  is the mandatory connectivity constraints for military terrestrial trunks.

When  $K = 5$   $GROUP(I,J,5)$  is the mandatory connectivity constraints for overall link.

Note: If  $GROUP(I,J,K) = 0$  then connectivity is not allowed.

If  $GROUP(I,J,K) = 1$  then always maintain connectivity.

If  $GROUP(I,J,K) = 2$  then allow for optional connectivity.

OFFERED LOAD ARRAY IN UNITS OF ERLANGS -  $OFLOAD(I,J)$  [Real]

$I$  = Source node.

$J$  = Destination node.

Note:  $OFLOAD(I,J) = 0$  for  $I = J$ .

OVERFLOW ARRAY IN UNITS OF ERLANGS -  $OFLARR(I,J,K)$  [Real]

$I$  = Switch being analyzed.

When  $J = 1$   $OFLARR$  is the mean class  $K$  traffic awaiting at node  $I$ .

When  $J = 2$   $OFLARR$  is the variance of class  $K$  traffic awaiting at node  $I$ .

When  $J = 3$  [ $K = 1$ ]  $OFLARR$  equals the class proportion factor. It tells the portion of satellite traffic carried terrestrially (class 1 traffic carried as class 2).  
 $OFLARR(I,3,K)$  is not used.

LINK STATISTICS ARRAY -  $LSTAT(I,J,K,L)$  [Real]

$I$  = Source node in a link.

$J$  = Destination node in a link.

When  $K = 1$   $LSTAT$  equals the mean of gross carried traffic.

When  $K = 2$   $LSTAT$  equals the variance of gross carried traffic.

When  $K = 3$   $LSTAT$  equals the mean of gross offered traffic.

When  $K = 4$   $LSTAT$  equals the variance of gross offered traffic.

When  $K = 5$   $LSTAT$  equals the traffic mean link blocking factor.

When  $K = 6$   $LSTAT$  equals the mean of rollback traffic.

When  $K = 7$   $LSTAT$  equals the variance of rollback traffic.

When  $K = 8$   $LSTAT$  equals the traffic variance link blocking factor.

When  $L = 1$  Link is the same as the satellite link.

When  $L = 2$  Link is the same as the terrestrial or military link.

#### LINK STATISTICS ACCUMULATOR - LSTATATA(I,J,K,L) [Real]

I = Source node in a link.

J = Destination node in a link.

When K = 1 LSTATATA equals the mean of gross carried traffic.

When K = 2 LSTATATA equals the variance of gross carried traffic.

When K = 3 LSTATATA equals the mean of gross offered traffic.

When K = 4 LSTATATA equals the variance of gross offered traffic.

When K = 5 LSTATATA equals the traffic mean link blocking factor.

When K = 6 LSTATATA equals the mean of rollback traffic.

When K = 7 LSTATATA equals the variance of rollback traffic.

When K = 8 LSTATATA equals the traffic variance link blocking factor.

When K = 9 [L = 1] not currently functional.

[L = 2] mean of gross offered class 2 traffic which is restricted from using satellite by ESB.

When K = 10 [L = 1] not currently functional.

[L = 2] variance of gross offered class 2 traffic which is restricted from using satellite by ESB [Not currently used].

When L = 1 Link is the same as the satellite link.

When L = 2 Link is the same as the terrestrial or military link.

NOTE: LSTATATA and LSTAT are compared to test for convergence during iterations.

#### LINK TRUNKS ARRAY - NTRUNK(I,J,K) [Integer]

I = Source node in link.

J = Destination node in link.

When K = 1 NTRUNK equals the number of satellite trunks.

When K = 2 NTRUNK equals the number of terrestrial trunks.

When K = 3 NTRUNK equals the number of military-owned terrestrial trunks.

Note: This array is assumed bidirectional and symmetric.

#### SWITCH CONSTRAINTS - SWITCH(I,J) [Integer]

I = Switch number.

When J = 1 SWITCH equals the maximum number of alternate routes.

When J = 2 SWITCH equals the routing protocol implemented at the switch.

When J = 3 SWITCH equals the maximum number of trunk groups which can be connected to the switch.

When J = 4 SWITCH equals the minimum number of trunk groups which can be connected to the switch.

When J = 5 SWITCH equals the maximum number of channels that can be connected to the switch.



#### NODE STATISTICS ARRAY - NSTAT(I,J) [Real]

I = Switch number.

When J = 1 NSTAT equals the mean of terminating traffic.

When J = 2 NSTAT equals the variance of terminating traffic.

When J = 3 NSTAT equals the mean of tandeming gross offered traffic.

When J = 4 NSTAT equals the variance of tandeming gross offered traffic.

When J = 5 NSTAT equals the mean of tandeming net carried traffic.

When J = 6 NSTAT equals the variance of tandeming net carried traffic.

When J = 7 NSTAT equals the originating mean poisson traffic.

When J = 8 NSTAT equals the mean rollback traffic with respect to NSTAT(I,5).

When J = 9 NSTAT equals the variance of rolled back traffic with respect to NSTAT(I,6)(J=6).

Note: Tandeming traffic is just passing through the Ith switch to its destination. So tandeming carried traffic is traffic originating before switch I and carried to other switches beyond switch I.

Note: Rollback traffic is either completely blocked by the network and rolled back to the switch or blocked at the switch itself.

#### POINT-TO-POINT STATISTICS ARRAY - PPSTAT(I,J,K,L) [Real]

I = Source node.

J = Destination node.

When K = 1 PPSTAT equals the mean of carried traffic.

When K = 2 PPSTAT equals the variance of carried traffic.

When K = 3 PPSTAT equals the traffic carried on primary path.

#### DATA POINTS DESIGN CONVERGENCE ARRAY - POINTS(I) [Real]

I = Iteration value (1 < I < 200).

Note: POINTS holds the mean end-to-end grade of service found at the end of the Ith iteration. If the POINTS array values converge before 200, then the program may be ended early. Otherwise, it will complete its iteration process. POINTS is used by Subroutine HUNT.

#### PRIMARY ROUTE CARRIED TRAFFIC STATISTICS - PRIME(I) [Real]

When 0 < I < 15 PRIME accumulates the traffic statistics for the primary paths of up to 15 tandemed links in length.

When I = 16 PRIME is used to aggregate a link weighted total of all completing traffic parcels. The formula used is  $PRIME(16) = \{ \text{Completing traffic parcel} \} \times \{ \text{Number of links used to complete traffic} \}$ . PRIME(16) is a function of PHIST(17,1,2). See Subroutine LOAD for details of its use.

## COMMON BLOCK COM1

WEIGHTED COST BETWEEN SWITCH PAIRS - WTCOST(I,J) [Real]

I = Source switch.

J = Destination switch.

Note: WTCOST(I,J) = 0 for I = J.

$$\text{WTCOST} = \frac{(\text{WTGSAT NSAT}) + (\text{WTGMIL NMIL})}{\text{NSAT} + \text{NPTT} + \text{NMIL}} \times \text{PTTCST}$$

Disallowed connections are set to a cost of 1.0E13. WTCOST is not assumed to be bidirectional. The array is used in Subroutine ISIZE.

TERRESTRIAL TRUNK COST ARRAY - PTTCST(I,J) [Integer]

I = Source node.

J = Destination node.

NETWORK DISTANCE ARRAY - KDIST(I,J) [Real]

I = Source node.

J = Destination node.

PREFERENTIAL ROUTING ARRAY - NPREF(I,J,K) [Integer]

I = Source node.

J = Destination node.

Note: This array is not currently used.

## COMMON BLOCK COM2

EMIN - Minimum point-to-point offered traffic in Erlangs [Real].

- Used when creating new pure satellite links. If point-to-point traffic summed bidirectionally is less than EMIN, then a new link may not be created. Calculated in Subroutine HIGOS.

NOITS - Number of iterations which will be made in any analysis cycle [Integer].

- Read in from input file. Set low for design mode; high for analysis mode.

MAXCHOP - Maximum number of satellite hops permitted in a traffic path [Integer].

- Read in from input file as 1 or 0.

NMOD - Number of nodes in the network [Integer].

BLOTM - Total mean traffic blocked by the network [Real].

BLOTV - Total variance traffic blocked by the network [Real].

APLOST - Total mean traffic given special handling during the loading process [Real].  
- This occurs because a traffic parcel becomes sufficiently small as to not warrant the computational overhead of loading it further.

PARC - Initial minimum size of traffic parcel in Erlangs below which normal routing will not occur [Real].

KPAC - Counter used to record the number of traffic parcels carried forward [Integer].  
- It is incremented each time traffic travels over an individual link.

LPAC - Counter used to record the number of traffic parcels rolled back [Integer].  
- It is incremented each time a traffic parcel rolls back over an individual link during the loading process.

MPAC - Counter used to record the number of individual traffic parcels reaching the destination switch [Integer].

MAXLEN - Maximum allowable path length parameter, measured in terms of the number of links in tandem [Integer].

#### COMMON BLOCK COM3

IN - Pointer to the input data file [Integer].  
OUT - Pointer to the output data file [Integer].  
LOG - Pointer to the running log file [Integer].  
GGPP - Pointer to the PPGOS.FIL file [Integer]. Not currently used.  
PPGOS - Point-to-point grade of service [Real].

#### COMMON BLOCK COM4

LTYPE - Preferred media type for links in adjusting connectivity and link sizing during design mode [Integer].

When LTYPE = 1	Preferred media is a satellite link.
When LTYPE = 2	Preferred media is a terrestrial link.
When LTYPE = 3	Preferred media is a military-owned link.

TINY - Minimum traffic parcel size (in Erlangs) allowed for PARC when reducing the special handled traffic [Real].

WTGSAT - Factor used in determining the cost of satellite transmission proportional to that of a terrestrial link on a multi-media link [Real].

WTGMIL - Factor used in determining the cost of military transmission proportional to that of a terrestrial link on a multi-media link [Real].

LBF - Initial value for each link's mean blocking factor [Real].  
- It is used in the first analysis iteration of each design iteration.

LVBF - Initial value for the variance of the blocking factor for each link [Real].  
- It is used in the first analysis iteration in each design iteration.

STABLE - Minimum value of the sum of the squares of the differences in link carried traffic between successive analysis iterations [Real].  
- It is used to measure the convergence so the iteration cycle may be stopped early.

TINYBF - Minimum value the link blocking factor may have before the program will automatically attempt to remove a trunk from the link during a design iteration [Real].

E1FRAC - Design objective for the distribution of the point-to-point GOS. It lies within EGOS of the mean PPGOS [Real].

E2FRAC - Design objective for the distribution of the point-to-point GOS. It lies within 2 EGOS of the mean PPGOS [Real].

DSNCNT - Maximum number of design iterations [Integer].

#### COMMON BLOCK COM5

SWITCH DAMAGE ARRAY - NOSWIT(I) [Integer].

I = Switch number.

When NOSWIT(I) = 1 Switch is damaged.

When NOSWIT(I) = 0 Switch is not damaged.

DAMAGED TRUNK ARRAY - NOTRNK(I,J,K) [Integer].

I = Source switch.

J = Destination switch.

K = Type of trunks between the switches I and J.

When K = 1 Trunk type is terrestrial.

When K = 2 Trunk type is satellite.

When K = 3 Trunk type is military.

USERLESS SWITCH ARRAY - NOUSER(I) - [Integer].

When NOUSER(I) = 1 There are no users at switch I.

When NOUSER(I) = 0 There are users at switch I.

## COMMON BLOCK ERLANG\_COMMON

IERLANGC - Determines which type of Erlang model is being used to analyze the network [Integer].

When IERLANGC = 0 Erlang B model is used.

When IERLANGC = 1 Erlang C model is used.

## LIST OF ACRONYMS

AT	Access Tandem
BSM	Bayesian Survivability Model
CDF	Cumulative Distribution Function
DCA	Defense Communications Agency
DEC	Digital Equipment Corporation
EMP	Electromagnetic Pulse
E.O.	Executive Order
GOS	Grade of Service
HEMP	High-Altitude Electromagnetic Pulse
IC	Inter-Carrier
K-S	Kolmogoro-Smirnov
LATA	Local Access Transport Area
LEC	Local Exchange Carrier
NCAM	Network Connectivity Analysis Model
NCS	National Communications System
NEN	Netherlands Emergency Network
NSDD	National Security Decision Directive
NS/EP	National Security and Emergency Preparedness
OMNCS	Office of the Manager, National Communications System
PDF	Probability Density Function
POP	Point-of-Presence
PSN	Public Switched Networks
PTT	Post, Telegraph and Telephone
QTCM	Queuing Traffic Congestion Model
VMS	Virtual Memory System

## REFERENCES

1. Traffic Congestion Analysis of a Circuit-Switched Telecommunications Network. National Communications System, January 31, 1989.
2. Development of PSN Traffic Modeling Capability. National Communications System, November 1989.
3. PSN Traffic Assessment Analysis. National Communications System, April 1990.