Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

COMMUNICATIONS MODEL FOR ANALYSIS AND DESIGN (COMMAND)

Brian E. Esterby, Larry J. Williams, and Mark H. Hellbusch
R & D ASSOCIATES
105 E. Vermijo, Suite 450
Colorado Springs, CO 80903

ABSTRACT

The COMmunications Model for ANalysis and Design (COMMAND) is a flexible and adaptable communications simulation model capable of modeling virtually any number of message source/destination and link combinations. COMMAND's flexibility is indicated by its data driven nature; most applications require database changes only. COMMAND is adaptable; if it doesn't contain the logic to support a unique protocol, buffer management technique, or other particular feature, one need only add or substitute a simple subroutine. Data structures and simulation control are hidden from and transparent to the user. COMMAND models traffic where message parameters vary widely, multimedia are employed, the user requires the capability to dictate dynamic environmental changes, and the content of the messages that reach their destination is important to measures of system performance.

INTRODUCTION

The COMmunications Model for ANalysis and Design (COMMAND) is an outgrowth of the communications portion of the end-to-end model of the nation's attack warning and attack assessment system. COMMAND is unique among its peers in that it effectively blends the importance of message content in a conventional traffic model with network disruption induced by external environmental stresses. In its original implementation, the impact of nuclear induced events on communications equipment and transmission media was the most important form of stress. In more conventional applications, any exogenous event, such as weather, other forms of interference, sabotage, or equipment reliability, can disrupt communications in deterministic or random fashions. COMMAND is very flexible and is entirely data driven for its many protocol, buffer management, routing, and media options. COMMAND is also easily adaptable to unique logic requirements.

From its beginning, COMMAND was designed to be data driven, modular, flexible, and adaptable. Those are not distinctive goals, nor is claiming their achievement. During the design, these attributes were assumed to be requirements for COMMAND to save the North American Aerospace Defense Command (NORAD) and other government users the expense and delay of procuring a new model every time they wish to investigate a change to their existing, rather inflexible communications system. The model either incorporates all foreseeable real world modifications, or it has established hooks for those that were intentionally excluded.

The data driven feature is implemented by having the user supply data to completely describe the communications network, message format and content, external stress, and the run control (e.g., file names and post processing reports) through a friendly preprocessor. Logic modules within the program are independent of the data structure and memory management. Thus, new modules can be added or substituted easily, if necessary. The model is adaptable to a wide variety of traffic flow analyses, not just traditional communications modeling. Parallel processor computer designs and automobile (or other commodity) traffic flow are just two examples that come immediately to mind. Finally, the flexibility of a virtually unlimited network size, efficient memory use and run time for a model of this complexity, and the option of deterministic or Monte Carlo operation round out the stated design features.

COMMAND, like all simulation models, consists of three fundamental parts: 1) an operational or real world system upon which the model is based, 2) a conceptual model which embodies the important relationships of the operational system, and 3) the computer model of the conceptual model.[1] If you are developing a general purpose simulation like COMMAND, there isn't a single operational system upon which to base the design, but a designer must appreciate the complexity and dynamic nature of any system the user will wish to model. The model designer must deal with the complex and changing nature of this undefined system when the conceptual model is formulated. The conceptual model is that set of understandings or perceived relationships which, to some degree of accuracy, represents an actual system. These relationships are then solidified into communications constructs, which form the building block structure out of which complex networks are built. The major constructs used in COMMAND are paths and channels.

A path is a single physical flow of information which contains only constant time delays and no routing decision making capabilities. A path can contain any number of communications nodes and links so long as there is no dynamic nature to it. The channel is a more abstract concept. A channel is a logical connection between a source of information (node) and a destination or set of destination nodes. A channel may have one or more paths emanating from it. Thus, a channel can be used to establish and control logical connections among nodes by the selective enablement of its paths. This selectivity process allows the channel to implement either static or dynamic routing algorithms.

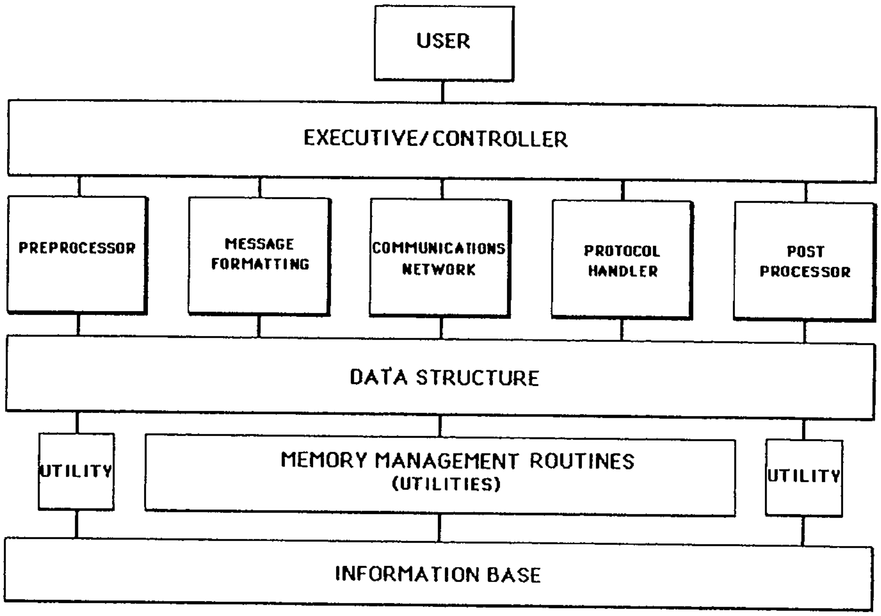

Figure 1: COMMAND Model General Structure

The combination of channels and paths creates a communications system topology. The user must understand this concept to be able to create a model of his network. The user must also realize that a topology can include the complex flow of information inside computer systems that are integral to his network as well as the flow information between the more traditional types of communications equipment.

Associated with each channel and path are message queues or buffers. Messages, not bits, are the smallest units of information which are treated collectively by COMMAND. Messages are stored in buffers until moved by some process associated with the channel or path. These associated processes may be related to the actions and characteristics of operational hardware, software or human activity.

Thus, the conceptual building blocks of COMMAND are the channel, path, associated buffers and related processes. A rich menu of processes allows the representation of an exceedingly complex system. The challenge, then, is to recast the operational system in terms of the basic building blocks of the conceptual model.

The computer model is the set of software and supporting databases which implement the conceptual model. COMMAND may be viewed as a set of software which forms the environment for the simulation and as sets of data which, when used by the software program, provide the specific character and attributes of the operational system being modeled. For example, the system topology, consisting of nodes and links and their connectivity, is in a database. Likewise, the processes used, such as path routing, buffer constraints and data link protocols, are identified in the database.

This report deals primarily with the computer software aspects of COMMAND. This emphasis on the software is not intended to diminish the importance of the conceptual model or the modeling process of a communications system.

DESIGN

COMMAND is an event driven simulation written in ANSI Standard FORTRAN-77. VAX FORTRAN-77 extensions are often used for data file manipulation to increase the processing speed, but the code is nominally machine independent. The diagram in Figure 1 shows the major modules integrated with the data structure and information base. This depiction emphasizes the modules as a collection of related subroutines rather than an actual flow of control. For example, the executive/controller is transparent to the user when he/she sets up a run with the preprocessor. The user doesn't directly interact with any other module. The assignment of a particular subroutine to a given module can be somewhat arbitrary.

Figure 1 shows seven major groups of software in addition to the parts labeled User, Data Structure, and Information Base. The following sentences briefly state the purpose of each module. More discussion will be presented in the paragraphs that follow. The executive/controller provides run control over the simulation by directly calling subroutines to initialize the simulation and managing an event calendar to execute appropriate logic modules. Logic modules are simply groups of related subroutines that are executed from start to finish to simulate a specific function. The preprocessor is that module with which the user interactively sets up his run, creating new databases or modifying existing files to achieve the required system description and run control. The message formatting module locates message format and content from input files and calculates message parameters affecting simulated transmission. The communications network module contains the subroutines to perform the actual movement of messages through the simulated system. The protocol handler module contains subroutines to execute and enforce the data link communications protocols over a communications path. The post processor module collects and analyzes the history of recorded events to produce summary reports of system and subsystem performance. The utilities module contains subroutines which are used throughout the simulation by other modules.

The executive/controller, unlike most of the modules that follow, is much more closely related to run control and execution than merely being a collection of subroutines awaiting a call to duty. In addition to run control, which is primarily the sequential calling of the correct logic modules, the executive/controller must manage an event calendar to order the calling of logic modules. The logic modules are structured so that once execution starts, the module can run to completion before rechecking the event calendar for subsequent events. Logic modules can create new events, they can terminate early due to events related to their own processing, but they cannot be interrupted to execute another logic module.

The executive/controller executes logic modules by first updating the system clock to the first or next event on the calendar. It then reads that first or next event, along with its logic module identifier and a pointer to the location of the data set used for this event. The identifier and pointer are both stored with the event descriptor. This event is removed from the calendar, and control is passed to the identified logic module. Between the execution of logic modules, the event calendar is reordered, since each logic module is allowed to write new events to the end of the calendar, regardless of the time of simulated execution, during its own execution. When no more events exist on the calendar or a predetermined end of simulation time has been reached, the simulation will stop.

The preprocessor is the user's interface with COMMAND. The preprocessor has been designed to be as user friendly as possible, but one should not imagine that an initial system description and run setup is a simple task. The preprocessor has been designed to use menus and other help functions, but it is the user's responsibility to be sure he understands the model's nomenclature and other idiosyncracies. In this area, the assistance of the model designer is essential. The preprocessor makes minor revisions to a simulated network very easy. These revisions are minor for COMMAND; they may be monumental for the operational system, like adding a new satellite and several ground stations to an existing land based network. As a final user friendly gesture, the preprocessor performs a network validity check to insure, within the bounds of what can be programmed into this function, that the network is indeed feasible and somewhat connected.

The user must provide information like the number and location of message sources, the number of outbound paths from these sources, the logical organization of the outbound paths, the location of intermediate nodes, and the type and performance levels of the connecting links. Other required information includes data rates, buffer capacities, protocol parameters, the effects of encryption or other coding schemes, and internal time delays. These types of information are commonly referred to and available, but the collection of this information into a single location may be a major task. The user may also select from a variety of post processing reports. Adding a new report to present collected information in a novel manner is a very simple task, i.e., a new subroutine.
The message formatting module reads the content of messages from the proper input file. Using a message index number provided through the preprocessor as a key, the format characteristics of the message are found from a sequential data file. This data file contains coded information describing the message fields in the operational system. The file also includes the length (in bits) of the transmitted message. The message is then passed to the communications network module for buffer placement and scheduled for further action by other modules. Lastly, the message formatting module reads the next record from the same input source. This record contains the time of the next formatting action, which is placed on the event calendar.

The communications network module is responsible for moving messages through the network. The communications network module has three different types of moving processes: movement from a starting node to a channel buffer, movement from a channel buffer to a path buffer, and movement over a path. These functions are shown in Figure 2.

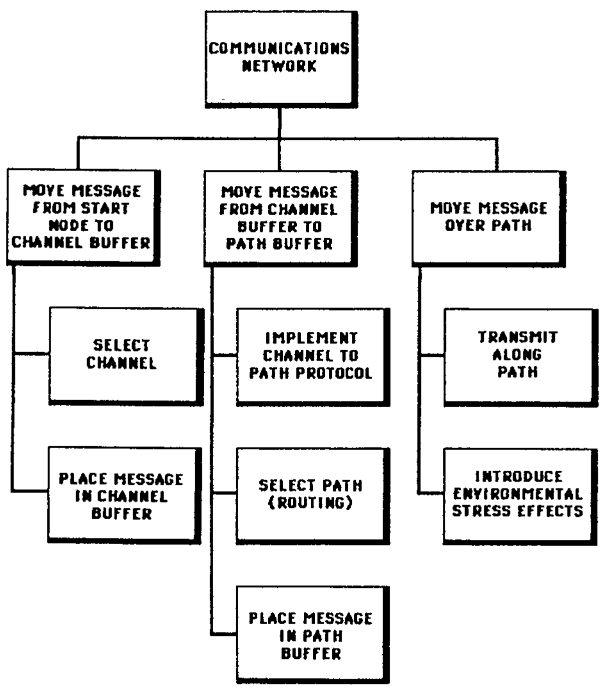

Figure 2: Communications Network Module

Movement from a start node to a channel buffer involves selecting the correct channel buffer and then placing the message in the channel buffer. There are many ways messages can be placed in the channel buffer depending on whether it is circular, non-circular, affected by message time-outs, hard priority partitions, soft priority partitions, internal channel multiple message blocking, or loaded from the top or bottom of the queue. The data set "CHANNEL" contains all necessary descriptions to control the proper placement of the message in the channel buffer.

Movement of messages from the channel to the path involves three processes: selecting the proper path, adhering to the transfer protocol between the channel and the path, and actually placing the message in the path buffer. The path selection process may be viewed as the implementation of routing decisions. Broadcast, primary/alternate, and trunked are static path selection subroutines. Special filtering algorithms based on the message index number are used in some applications. Dynamic real time routing selection algorithms, like those used in the ground wave emergency network (GWEN), are incorporated. Adherence to the protocols between a channel and a path involves consideration of when messages may be transferred to the path, the conditions under which messages can be deleted from the channel buffer, and preservation of required message blocking. Like the channel message buffer placement, message placement in the path buffer has similar options and constraints. Parameters controlling the process selected are contained in the "CHANNEL" and "PATHINFO" data sets.

The movement of a message over a path is conceptually simpler in operation than either of the two previous movements. The message (or acknowledgment) is progressively moved from node to node along the defined path. The determination of successful transmission is made at each link by choosing a random variable and comparing it to a threshold value. The threshold value is determined by subroutines containing the environmental effects of the ambient or stress conditions. Subroutines determining ambient error probabilities have been developed from empirical data. Unique stressed environment routines are also available to account for nonstandard conditions. The user has the choice of statistically modifying the ambient environment calculations or implementing a special environment and probability calculation.

The protocol handler module implements the defined communications protocols for the network. Protocols define how the system works by providing the rules and procedures the system employs. These protocols describe what to do when a message is ready, where to put a message that is received, how to respond to an errored message, how long a message can be retained before it is discarded and many other details essential to the communications functions.

The protocol handler module performs two main tasks. First, the protocol determines the appropriate action to be taken. The collection of logic flow for a particular protocol is frequently referred to as the state diagram, since the current state of the protocol uniquely defines the exact actions with which the system must comply. Examples of the types of actions identified include scheduling a retransmission, deleting a message from a buffer, and updating the state of the protocol device. Second, the identified actions must be performed. In COMMAND, this may be done in three ways: direct and immediate simulation, scheduling an event on the event calendar, or calling a utility module to perform the necessary action. Direct actions are those which are totally internal to the protocol itself, such as updating selected timers or counters. Scheduling events on the common event calendar includes such decisions as the protocol scheduling itself for recall at a time-out or scheduling transmission of a prepared acknowledgment. The implemented protocols include broadcast, Autodin Mode 1, several variations of ADCCP (Advanced Data Communications Control Procedure), and the GWEN-unique protocol. This variety includes the logical structure to describe most common protocols. Other logic submodules for protocols will be written and permanently incorporated as they are needed.

The post processing module is divided into two sections. The first performs data collection during the simulation. The second performs the analysis of the collected information and prepares the reports in a comprehensible form. In addition to standard reports that focus on the network description and summary statistics, other reports are available that will concentrate on user directed areas of emphasis. The prepared report formats concentrate on measures of message throughput, either aggregated, by specific source and destination pairs, or by specific message type. Buffer capacity and utilization reports are also available. If all else fails, detailed event history listings are available.

The utilities module is a collection of subroutines used by more than one of the other major modules. These routines are collectively referred to as utilities in a separate module, rather than repetitively referring to them in more than one module description. A schedule interfaces submodule performs the event creation activity. A buffer housekeeping submodule includes subroutines to manage all actions associated with both channel and path buffers, including message insertions, message deletions and time-out monitoring. Single and double linked list functions allow the functional subroutines to manipulate and control information within the data structure. The memory allocation subroutines provide data set memory for the functional subroutines. Because of the large number of different data set types and the profound consequence of memory location errors, the memory allocation module provides exhaustive type checking of all fields including memory location pointers.

The data structure is an internal strength of COMMAND. Figure 3 depicts the data structure for channel and path information and the particular relationships among the myriad of information that must be stored and manipulated in most simulations. The data structure is based on variable length record size with mixed variable types and pointers between the records. Among other benefits is a capability to use Pascal-like linked lists with pointers to successor and predecessor records. These features have been implemented with utility subprograms written in FORTRAN. The data structure provides the flexibility to describe all network configurations efficiently in the available memory. The user need know nothing about the data structure. A systems analyst adding or modifying a subroutine need only know and understand the data structure of

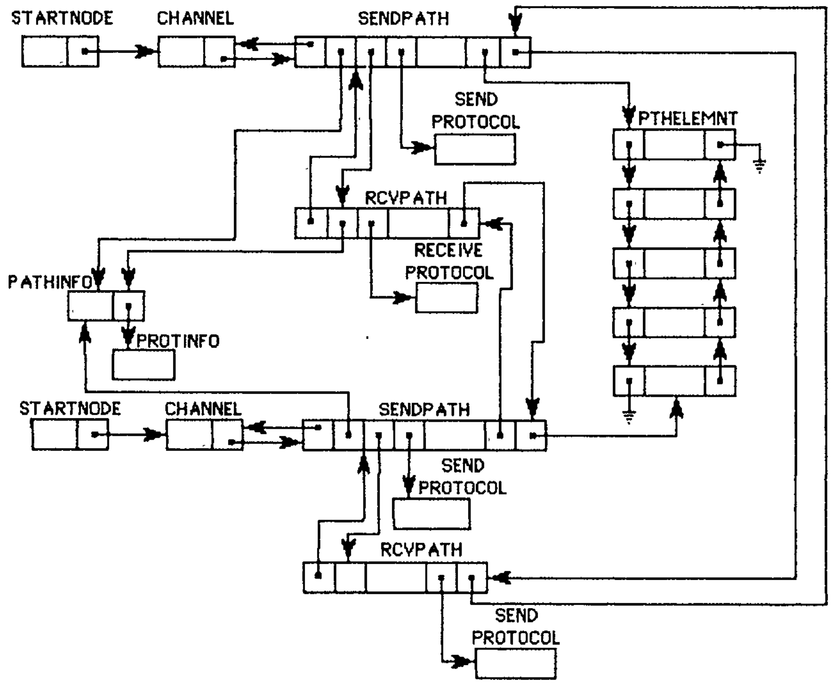

Figure 3: Data Structure for Channels and Paths

the variables and their associated pointers with which he/she is dealing; he/she need not modify the program's memory management routines. In this sense, COMMAND extensively employs both data hiding and data abstraction.

The information base in Figure 1 refers to the use of memory within the particular machine on which COMMAND is running. Again, FORTRAN utility subprograms dynamically allocate and manage memory utilization. These routines periodically compress the memory used after history events have been written to post processing files and pockets of random access memory have been freed.

PERFORMANCE

COMMAND has been designed to execute a scenario with about 10 message sources, 5 destinations, 275 nodes with connecting links of several media types, severe stress requiring constant recalculation of environmental conditions, and 40,000 messages, with minimal data recording (default post processing reports) in 30 minutes on a VAX 11/780. This speed is less than a factor of two times real time. Exercising some or all of the post processing report options will lengthen the run time. Less complex scenarios will require much less processing time.

EXAMPLE

The first uses and analyses with COMMAND have emphasized the message generation part of the communications process. Message generation is part of COMMAND, unlike other models which begin with complete messages. Message generation begins with an external stimulus and data input, such as the output calculations of a radar sensor looking for threatening objects in its field of view. The sensor software assesses the threat and determines which one of a prescribed set of message types should be sent to headquarters. COMMAND acts on this stimulus at the specified simulation time by formatting a message, calculating parameters that will affect transmission success, and attempting to simulate the transmission of this message to the next node. If a communications backup exists due to heavy traffic or a down line, the new message must be stored or otherwise disposed of as the real system would. The process of generating the message and moving it from a sensor computer to the communications equipment is a microcosm of the whole communications process from source to destination. For simplicity and brevity, only the limited message generation process will be described in the paragraphs that follow.

Message flow starts inside the sensor computer and proceeds under both software and hardware control through an internal computer communications network to the point where the message is ready to physically leave the sensor site on its way to a command center. The simulated formatting and local transmission (computer to communications equipment at the site) of messages are implemented in COMMAND by designated Logic Modules (LMs). Each logic module describes a non-interruptible sequence of events(Figure 4). LMs must be executed in a relevant order to model the message generation function. The order of processing is determined by data in user supplied configuration files.

LM-A reads message variables from a particular sensor file and reformats the data values according

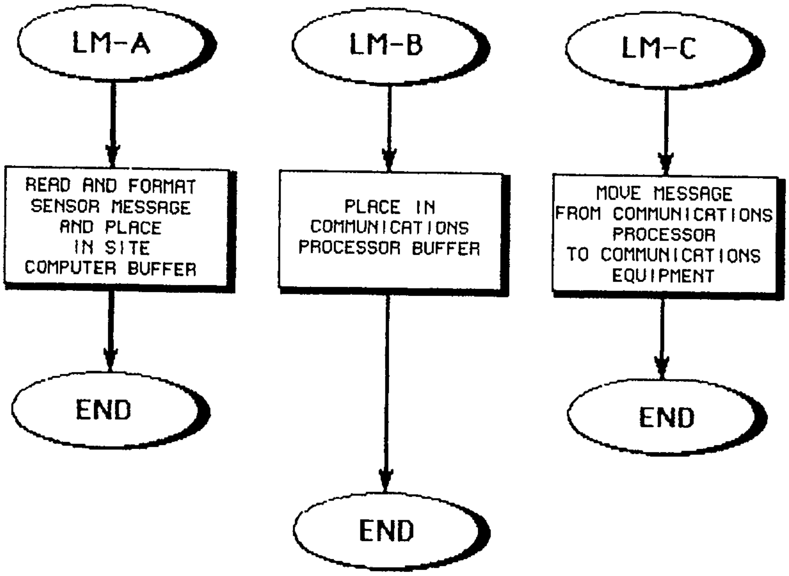

Figure 4: Logic Modules Used for Message Generation

to the bit patterns in the specified message. LM-A also places a message pointer reference to the sensor computer buffer and attempts to move the message into the communications computer using LM-B. LM-C transmits the message in the broadcast mode from the communications computer to the communications equipment. If this transmission requires ADCCP protocol with an acknowledgment of message receipt, other logic modules are substituted for LM-C.

Figure 5 shows both the top-level message formatting actions contained in the LM-A and the fact that logic modules can use subroutines from different modules of COMMAND. The message formatting module reads the event time of the first message and places a sensor message generation call on the event calendar. Upon entry into the logic module, subroutine "AMSFMT" selects the field of control codes from the message format file. A data set is created to store the forthcoming message. Next, the actual message data is read from the sensor message file. Using previously read formatting codes, the numerical message data are rounded to fit the prescribed bit pattern and placed in the waiting data set. A history event is recorded, documenting the message creation.

Subroutine "ASTRPL" begins the simulated movement of the message to the computer buffer and schedules LM-B to move the message into the communications computer buffer at the appropriate time. Each computer has its own buffer size and management scheme. Figure 6 summarizes the buffer transfer logic of subroutine "ASTRPL".

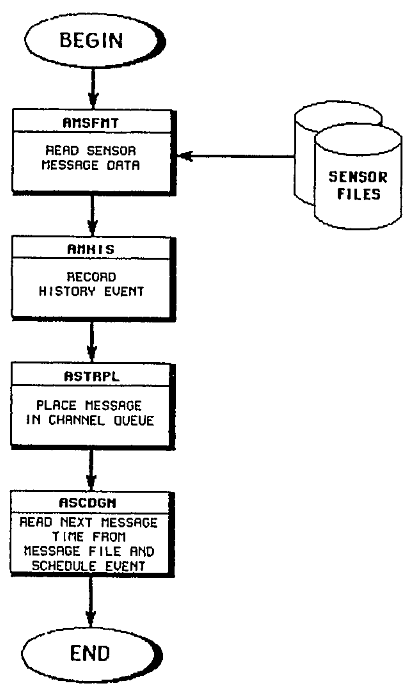

Figure 5: Flow of Logic Module A

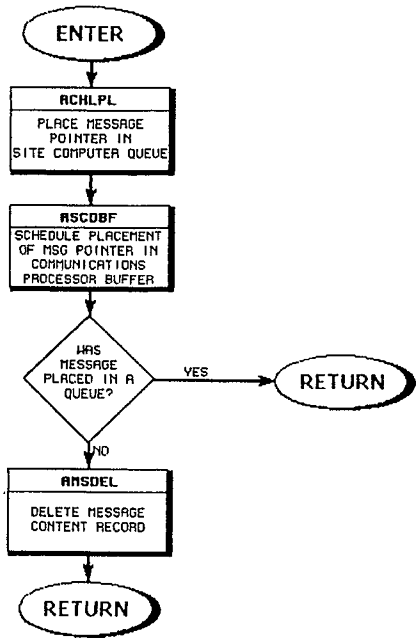

Figure 6: Subroutine "ASTRPL"

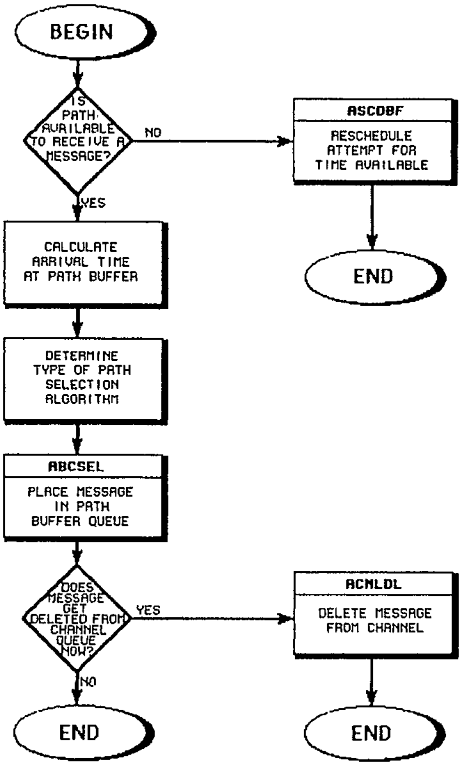

Figure 7: Flow of Logic Module B

Buffer to buffer transfer is implemented by LM-B, Figure 7. The path between the site computer and the communications computer is checked to insure that no other message is currently being transmitted on the channel. If a message is being transmitted, the completion time is noted and the logic module is rescheduled; otherwise the transmission proceeds. The path selection aspect of LM-B selects the appropriate output port according to the prescribed logic for the site. Message flow is controlled by the status of the buffer in the communications computer. If the buffer is not full, the message is accepted and transmission to the communications equipment is scheduled.

The logic of LM-C is shown in Figure 8. Upon execution, the message is sent from the communications computer buffer by the subroutine "AMGTRS". A timer is also available in "AMGTRS" for applications involving accountability, like ADCCP. Possible loss of function of the sensor site is checked through stress update events. The message is finally sent to the communications equipment. Subroutine "ASEND", which is the heart of "AMGTRS" and is shown in Figure 9, controls the actual sending process. Arrival of the message is recorded as a history event from "AMGTRS" after the return from "ASEND".

The message just received by the communications equipment can now be deleted from the communications computer buffer, leaving at least one available buffer position. Subroutine "ASCDBF" queries the site computer to identify the next available message. If a message is available, LM-B is scheduled to transfer it to the communications computer buffer.

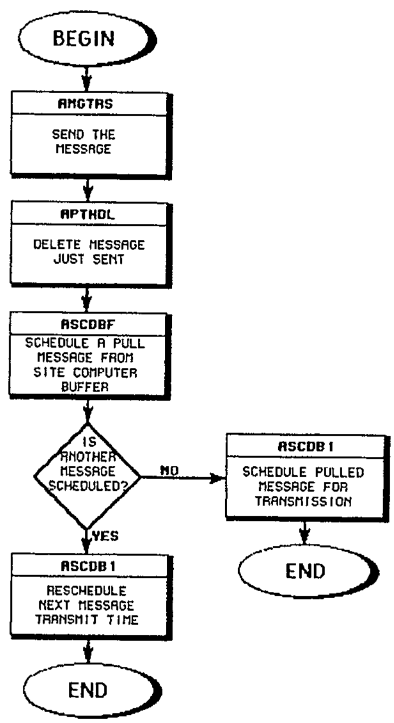

Figure 8: Flow of Logic Module C

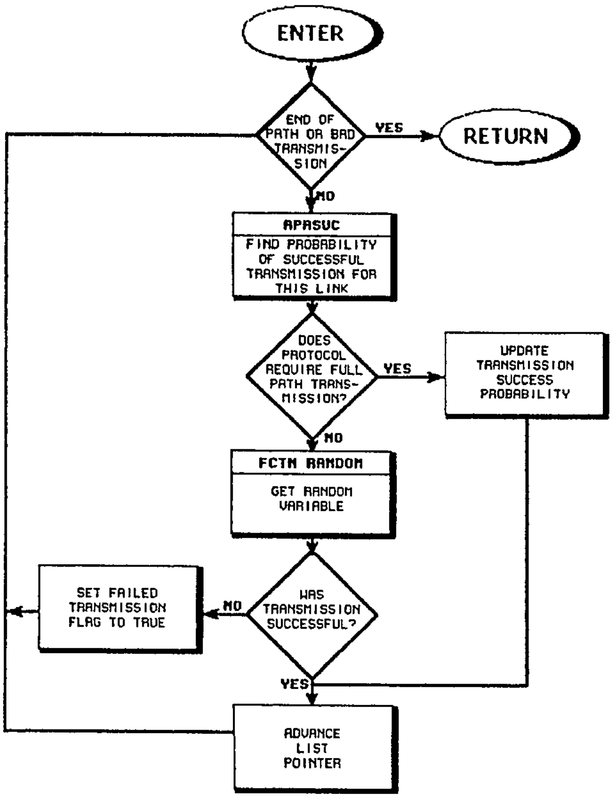

Figure 9: Subroutine "ASEND"

Long haul communications processes are very similar to those described above. Some networks have a dynamic routing scheme that invokes a special set of logic modules. The most important task for COMMAND is simulating the message accountability of ADCCP or another similar protocol. Acknowledgments are simulated. The communications equipment at the source disposes of messages when acknowledgments are received and resends the message after an appropriate time if there is no acknowledgment or an out of sequence acknowledgment is received.

The post processor develops reports in several forms: tables, charts, matrices, and text. An example is shown in Figure 10, which depicts the time history of message generation events during a completely fabricated test scenario from a radar site. Reports such as this are useful for finding system bottlenecks and understanding the demands on the network's processing capability.
Figure 10. Message Generation Example Report

SUMMARY

Command was originally designed for a very specific purpose, but the resulting model has evolved to a broad range of capabilities. The mixture of performance based upon message content, sophisticated protocol handling and buffer management, dynamic path switching and circuit establishment, and complex link reliability modeling (stress) appear to be a unique combination of features when compared to similar models.

REFERENCE

1. Hoover, Stewart V., and Ronald F. Perry, "Validation of Simulation Models: The Weak/Missing Link," *1984 Winter Simulation Conference Proceedings*, 1984, pages 293-295.

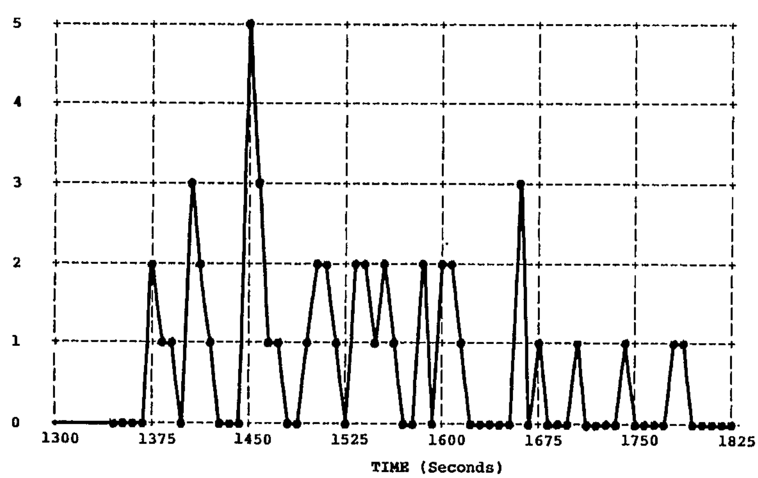

Figure 10: Message Generation Example Report