MIT/LCS/TM-127

A NETWORK TRAFFIC GENERATOR FOR DECNET

Richard J. Strazdas

March 1979

MIT/LCS/TM-127

A NETWORK TRAFFIC GENERATOR FOR DECNET

by

Richard J. Strazdas

January 19, 1979

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

A Network Traffic Generator for DECnet

by

Richard James Strazdas

Submitted to the Department of Electrical Engineering and Computer Science on 1979 January 19 in partial fulfillment of the requirements for the Degrees of Bachelor of Science and Master of Science

ABSTRACT

Computer network traffic generators provide a means for supplying benchmark results and for measuring computer network performance at all levels. Eventually they will also aid in fault diagnosis. The network traffic generator described in this thesis allows flexible yet convenient control over a number of parameters useful for generating loads over both test and real networks based on DEC's PDP-11 minicomputer. Implementation on a test network provides sample results. A discussion of design compromises, and recommendations for further study and design point to various open issues.

Thesis Supervisor: Prof. Liba Svobodova

ACKNOWLEDGEMENTS

Many people deserve much thanks for their involvement in this paper. My hat is off to Mr. Ralph DeMent and Mr. James Hirni for providing me with the time, space, and equipment at DEC to perform this research. I bow to Professor Liba Svobodova whose patience and helpful criticism were a constant guiding light. A hearty salute goes to Mr. Christopher Hume who (almost) never tired of helping solve my implementation problems. And a round of applause is dedicated to my coworkers, cohorts, and companions for making the work light. Finally, I reserve my deepest gratitude for my parents who are my greatest source of inspiration. It is for them more than anyone else that this is written.

TABLE OF CONTENTS

I. Problem Statement and Motivation

An important issue in the development of any service, and in this case the development of a computer network, is performance: How well does it work under certain conditions? Will it meet the user's requirements, the designer's specifications? Consequently, much effort has been put into defining performance criteria and performance measurement, and into developing tools that facilitate the gathering and reduction of useful measures from the massive amount of data flowing through a computer network.

A key concept that has emerged is the network control center, a centralized network host where monitoring, measurement, and control of the network take place. (The measurement function can be performed at a separate specialized host, as is done in the ARPANET, but the network configuration on which this research has been developed conglomerates all three functions in one host for efficiency in hardware and personnel.)

One of the primary responsibilities of the control center is to ensure performance standards, as protocols, routing or flow

control are modified, or new hardware introduced. Benchmark results are the most direct means of answering the questions posed in the beginning paragraph.

The way to accurately obtain these results is to monitor "live" data flowing through the network. One would think at first that the data should also be "real", but two problems come up. Firstly, user generation of data is haphazard, varying enormously from minute to minute, so overall performance loses meaning as it bears little relationship to individual use of the network facility. Secondly, under conditions of network development, there is no user community and therefore no real data. The solution is, of course, to emulate real traffic by generating it according to a number of parameters specified by the experimenter.

In its simplest form, that is precisely the goal of this study: to design and build a flexible generator of network traffic in order to ease such tasks as testing routing software and load capacity, and establishing basic performance ratings such as throughput, delay, efficiency, and reliability.

II. Relevant Work Done

The literature specifically concerning artificial message generators is understandably rather sparse. Details are left to internal documentation, while it is typically left to papers dealing with network measurement, performance, or testing to mention generator functions and how they relate to the rest of the performance/measurement software or hardware. This is not a serious drawback, however, since the programming of the generator, while far from trivial, is hardly state-of-the-art. The functions to be supported and the network interface and user interface designs are the really crutial decisions affecting the traffic generator's incorporation into the network. Thus, my initial study of this subject led me to descriptions of network measurement centers, monitoring machines, and the like.

As may be expected, my reading began with the ARPANET. ARPA's Network Control Center, situated at the Bolt Beranek and Newman node, is used to "operate, maintain, and modify the communications subnetwork", which includes statistics gathering. The Network Measurement Center, located at the

UCLA node, uses measurements from real traffic, and from tests using artifically generated traffic, primarily to verify and improve analytic models of network behavior. Each node (IMP) is capable of sending messages to a single destination over a single link with the following parameters: length (fixed), interdeparture time (fixed or driven by return message), trace on or off, and the ability to send to a nonexistant host - a bit bucket. (Tracing, when enabled, causes timing information to be sent from each node through which the message passes.) Additionally, some hosts have multiple pairs of generators and acceptors, and there are programs at UCLA to format the messages which will initiate experiments.

The National Bureau of Standards has a Network Measurement Machine (NMM) which is essentially a minicomputer-based probe with mass storage. A hardware character generator with a few inter-character and baud rate settings will send a specific character or entire character set in sequence to the NMM. As it stands, however, it cannot fulfill the requirements of a message generator.

Much work has been done on network monitoring at the University of Waterloo in Ontario. The software for their Network Monitor Center is highly modularized, resulting in clearly defined functions for each of the manager routines. Of particular importance is its communications manager which is responsible for coordinating communications between remote

monitor centers, the load generator, moniotring software, and data analysis. The load generator emulates up to sixteen terminal users, and will soon also generate higher speed (host) traffic. In addition, it transmits prepared scripts.

General Electric has also been a pioneer in this field with work on its Network Monitor System. Its monitoring strategy is roughly similar to that of ARPANET, while its strongest point is its ability to display timely status and statistical figures in a variety of useful summary formats. This, in conjunction with automated alarms, can significantly advance the time when approaching problems are recognized. I could find no reference, however, to a traffic generator.

Relevant work on load generation at Digital Equipment Corporation (DEC) began approximately three years ago. In order to test DECNET on the PDP-8, a "dynamic exerciser program" was written. It serviced up to ten logical links (to receivers) where the receiving task mirrored the random data sent. Parameters used were destination nodes, message length, delay between messages, and connect period, which is essentially how many messages are sent before the logical link is broken and another one established. Another interesting feature was noise generation; this can be extremely useful in a test environment where all the nodes are in the same building or room, for simulating more unreliable communication lines.

A preliminary performance evaluation project later wrote a message generator that specified destination node and program, and number of messages and their length. This was used primarily for CPU usage and throughput measures on PDP-8's.

A specification for a more detailed DECNET performance evaluation calls for a message generator that will produce messages of fixed or exponentially distributed lengths and interarrival times to up to eight tasks at various nodes. Mesages may be sent independently of previous replies or even as fast as the protocols will allow. Control will be from an "experiment control point" at a designated node.

There are other systems that transparently monitor a network's traffic via probes, but these do not inject messages into the network and so are not considered here.

III. Design

This chapter describes the ingredients of a general-purpose computer network traffic generator: the characteristics of the possible traffic patterns, the information to be collected and recorded during an experiment, and the modular construction of the generator that provides the needed flexibility.

No attempt has been made to pin down the features of the complete do-it-all traffic generator. Indeed, not all the functionality described here is included in the present implementation. (This is due primarily to time constraints. Omissions and recommendations are discussed in the following chapter.) However, it is felt that no important feature has been left out of the design, and that it should suit all but the truly esoteric needs of performance analysts.

The components of the traffic generator system are described in detail in section C, but a few words here about the system's structure will help keep the intervening discussions in the proper perspective. Instructions flow from the control node to individual nodes, and thence to the generators and

other programs in those nodes. Each generator establishes a logical link, or connection, to the receiver in the traffic destination node specified for that generator, and passes trafic over this link. The generators, the receiver, and other programs record information pertaining to that link and to the node in general. Section A describes the parameters that define the traffic pattern of each link, and section B describes the data that is recorded.

A. Parameters for Description of Test Traffic

Apart from the ability to perform its stated function, the power of the traffic generator lies in its flexibility. The experimenter, in order to aid his understanding of a network's behavior, must be able to monitor the network's performance in several modes of operation, including diverse traffic conditions and various configurations.

The following are the parameters essential to accurately describe a traffic pattern on a network (one set for each emulated network user):

source node

destination node

message length: average value, exponential parameter

intermessage time: average, exponent

time base: send next message immediately, after acknowledgement, or after reply

passive or active receiver

messages between connects (number of messages per session): average, exponent

In this section, I shall explain the reasoning behind each choice of experiment description parameter.

Most important, of course, are the source and destination nodes for the traffic paths. When describing the experiment, each generator is associated with a source node, and a corresponding receiver is associated with a destination node. These node names are stored in each record of the experiment description table. This table is preprocessed, using these two fields as keys, to determine which records should be sent to individual network nodes for local processing.

The second important parameter is the length of the messages sent. There are two ways in which to approach this, and this project provides for both. The first option is to have all messages of the same length. This is trivial to implement. The second option is to allow variable length messages. There are naturally many ways to do this, but the one that stands out clearly is a message length with an exponentially distributed probability mass function. There are several reasons for this choice: it is easy to implement, it provides a great deal of variability, and most importantly, it ideally suits modeling and simulation studies since it represents a

Poisson service time distribution, the second 'M' in the M/M/n queue which is the mainstay of network modeling.

Deriving an exponentially distributed random variable from a uniformly distributed random variable is not difficult. The probability mass function is described by:

$$y = p_x(X) = ae^{-aX} \qquad (1)$$

Using the classical inversion method for deriving the new distribution, we obtain:

$$X = -\ln(y)/a \qquad (2)$$

Finally, using the user-specified parameters AVE for average value of the random variable, and EX for the exponent (a, in the above equations), and noting the fact that the expected value of the length L is 1/a, we arrive at:

$$L = AVE + (\ln(1/random) - 1)/EX \qquad (3)$$

This yields the desired distribution, with a minimum value at AVE - 1/EX. By rounding to the nearest integer, the message length is obtained.

The next parameter is the intermessage time. This is the real

time between sending successive messages, and is used to emulate processing time between intertask communication. This was approached in the same way as the message length, with the user-defined average and exponential parameters, except that no rounding is necessary as time is real-valued.

The generator task issues an executive call to mark the passage of the intermessage time span, and then goes to 'sleep'. In this way the generator is awakened just in time to send the next message.

An important factor in traffic definition is whether the receiving task talks back to the message generator, or acts as a bit bucket, absorbing all messages. This of course may make a difference of up to a factor of two in the traffic between the nodes. In talk back mode, the receiver will act as a mirror, sending back the same message. (There is little to gain by defining a different return pattern.) This parameter is used in conjunction with the following one to fine-tune the traffic to the desired properties.

The base on which the intermessage time is to be measured is offered in three options. One can have the generator begin waiting the allotted time after it receives a reply from the destination task (which means that the talk back parameter must be set). The generator can also wait beginning with the

acknowledgement of its previous send command. Finally, it can simply not wait at all. The first option characterizes a dialogue between the two communicating tasks, the second characterizes bulk information transfer, and the last enables one to test the capacity of the network path between the two nodes.

Traffic in a network is further constituted by the durations of the dialogues themselves. To this end, the experimenter may specify the number of messages to be sent in each session, in a manner entirely analogous to the message length. This can be used for tailoring the experiment if the intended application(s) characteristics are known, or for making the communications as diverse as deemed necessary. In either case, this is important because there is much overhead associated with connecting (establishing) the logical links (network paths) and disconnecting them, and the network's behavior is obviously affected by the extent to which this takes place.

B. Measurements of Test Traffic

Just as the experiment description parameters were chosen to provide flexible but concise control of the traffic generation, so equal care must be given to the choice of data available after an experiment and to the presentation of this

data. The keywords here are "performance measures" and "user interface".

It is noteworthy that this research specifically avoided the latter issue. Output presentation is aimed at data interpretation and this is highly dependent on the nature of the analysis taking place. For example, periodically conducted experiments yield data that, when combined with previously archived results, can clarify trends due to software enhancements, sunspot interference, or almost anything else. Or statistics can be combined with those of an experiment with slightly perturbed inputs to gain insight into performance degradation, etc. Since it became clear that the needs for formatting results were as diverse as the experiments that could be performed, my data presentation is simply a structured dump of the data collected. (An example can be found in Appendix A.) Consequently, this chapter will be concerned only with the choice and description of those data.

The data collected and recorded can be divided into three classes. The first deals with administrative information, and can range from the time of the experiment to the state of the network to statistics on the values that were assigned by the probabilistic experiment parameters. This class might be divided further into two areas: one documents the circumstances of the experiment and the other verifies that the probabilistic variables behaved as prescribed.

The second class provides summary information. This includes counts, averages, and second moments, and describes the behavior of certain logical and physical units on a gross level.

The final class is concerned with detailed information on individual entities. This can be the shortest, the longest, the first, the last, every fifth one, etc. In this way, extreme cases as well as typical ones can be examined in detail.

For completeness, one might like to include a fourth class, derived data. But these data are really implicitly contained in the other classes.

1. Administrative Information

Because very few experiments will be performed in vacuo, and data may be analyzed months after it is collected, several pieces of information must be recorded simply to ensure that the circumstances around the experiment are not forgotten. These 'measurements' are:

date and time of the experiment
experiment parameter values
network configuration for the nodes involved

The last piece of information is not necessary unless the network is very dynamic; one can always check the configuration from records kept by the network's administrator. In

addition, the principal experimenter's name and a field for commenting on the experiment's purpose can be maintained.

Also of interest is whether the probabilistic parameters cited in section A lived up to their prescribed behavior. After a long experiment this will be nearly certainly true, but under other circumstances it would be reassuring to be able to confirm this. Therefore, the following statistics are gathered:

message lengths: number of, sum of, and sum of squares of

intermessage times at sender: number of, sum of, sum of squares of

messages per connect: number of, sum of, sum of squares of

The three statistics of each item are sufficient to calculate the mean and variance.

2. Summary Information

Ideally, one would like to collect every detail about every message sent in the network. Unfortunately, computer memories were not made to be used so frivolously, and transferring this information would about double the load on the network, so a more reasonable approach must be designed. (Besides, the data would still have to be processed for human consumption later in order to be studied effectively.)

There are two main solutions to this problem. One involves gathering histogram data: the frequency of an event or any other value that may fall within any of a specified set of boundaries is recorded. From this one may draw a bar graph or make statements about the statistical conduct of what was measured. The second solution requires only a few statistics to summarize the behavior of the measure of concern: a count, a sum of values, and a sum of squares of values. One could also implement both solutions since the data collected are not totally redundant.

While a histogram retains a lot of the information in a relatively compact form, the level of detail is still too great for most applications. This is partially manifested in the fact that choosing the boundaries is a problem that must be solved interactively and not without difficulty. The approach that was chosen was the simpler one of gathering just enough data to be able to calculate the first two moments; this and the information described in section 3 below allow many important statements to be made about the network's performance.

The major tool that is being sacrificed here is the ability to produce a distribution curve. However, this would require a histogram of very fine resolution and should be relegated to a special-purpose program.

The data that falls into the summary category all relates to timing. For each message, the following variables are used to update a count, a sum, and a sum of squares.

intermessage time at receiver

one-way delay

round-trip delay, if appropriate

The first measure must of course have an average equal to that of the intermessage time at the sender; comparing the variances will yield useful understanding of such network operations as buffering and queueing. The latter two measures are probably the most important ones that can be made, for they are the values that are of greatest importance to the user. This of course assumes adequate network availability and accuracy.

3. Detailed Information

This class of information concerns individual message details, transport errors and extreme cases, as well as miscellaneous items.

Let us first discuss what is meant by 'message details'. It is very desirable to be able to follow a message through its entire life, recording where it was, how long it took to get there, and how long it stayed there. This should take place not only at the node level (useful for routing verification

and statistics) but also within the node (for details on queue delay, retransmission, etc.). This capability is called tracing and generates the following data at each node and for each message:

message header (identifier)
time of arrival at network software
time ready for output
time transmitted
time acknowledged or put on retransmission queue
output line number

An intermediate node (one that is neither the source nor the destination) would record all this information. An endpoint node would of course gather only the input or the output statistics as appropriate.

It should be clear that this ability requires modification to the network software itself; the rest of the traffic generator is designed purely as a network application and has no access to the internal message processing. This function also revives the problem of the extent to which one is willing to commit resources to collect all this information. And finally, the experimenter must be able to select which messages should be traced. (This would have been discussed in section A but for the fact that tracing was more properly introduced here.)

To begin, there must be a mechanism for the network user to specify whether a given message is to be traced. This first modification is easily accomplished by a special argument to the SEND call which would cause a trace bit to be set in the header of the message. This feature could however be abused since it may be utilized by any user. A network control command to enable tracing at specified nodes will alleviate this problem. Turning on tracing just before an experiment, and only at those nodes where it is useful to do so, will eliminate a mass of useless overhead. As a message is processed within a node's network software, a small buffer is temporarily attached to it and updated as the message moves from one stage to another. As soon as the node can dispose of the message (after it has been transmitted and acknowledged or passed to the end user), this buffer is transferred to a program (the data updater, described in section C) which appends it to an area reserved for such information.

The size of that area is one of the parameters that the experimenter must specify, as well as which nodes should be trace-enabled. This is important because trace data can quickly fill a lot of space and, rather than transfer this data online to the experiment control node (from where the experiment is being conducted) and risk perturbing the very traffic being measured, the threshold of a full trace area will trigger a trace disable at that node.

The last topic on tracing is describing which messages sould be traced. The experimenter can specify, on a per-record basis in the experiment description table, the frequency or relative position of the messages to be traced. For example, one may trace the first message after each connection, or one may trace every tenth message. Dynamically arranging tracing (e.g. tracing a message that has had to be retransmitted more than once) appears to be another useful feature but this has not been researched here.

The rest of the detailed information will follow quickly since it is simpler to describe than tracing. The number of connects and disconnects for each user path (source--destination pair) are collected here; these values reflect communication overhead. For each path, maximum and minimum values are recorded for each of the three timing measures in the summary class; these indicate extremes of performance. The CPU utilization at each node is measured by running a lowest-priority program that loops forever, incrementing a counter. Knowing the counter's rate of incrementation and the duration of the experiment makes calculating the fraction of idle time a simple matter.

Another area of useful information is status information that can be gotten for "free", with an existing command to the network software. In this case, one can retrieve the active-/inactive status of all nodes, error counts for all nodes and

for all links, and a few other goodies that are uninteresting for our purposes. This command can be executed periodically at the experiment control node where storage should not be a problem. If experience shows that high-frequency use of this command causes data overflow on long experiments, then a threshold mechanism similar to that for tracing can be implemented.

The final desirable feature of traffic measurement complements the preceding one: other status information can be retrieved via a programmatic interface to the network data structures. This can take the form of either additional user commands or special interfaces which can be accessed directly by the application program. The periodicity and storage arguments here are the same except that, since the data will be collected locally, a threshold will certainly be needed. These data are:

free buffer length
each modem's output queue length
sent queue (waiting for ACK) length
retransmission queue length
conversion queue length
store and forward queue length

It should be mentioned at this point that at the end of an experiment, all the data collected at every node is sent to

the experiment control node for storage and analysis.

C. Program Modularization and Experiment Control

As with any complex, distributed software project, proper program modularization is the key to nirvana (or at least to its closest approximation in a computer environment). This section will detail the division of function among the on-line modules as well as their control structures. Various other separate (off-line) but necessary programs will also be discussed here.

The reader should be aware that a few parts of the following discussion deal with software features specific to the RSX-11 environment. Where possible, all such references have been relegated to the appendices, but discussion of the realization of certain design goals requires that they be mentioned here.

Before describing the traffic generator's organization, it would be useful to define the assumptions that have been made and the philosophy of its use.

The first point to note is that the whole system runs as an application, on top of the network. This naturally has its disadvantages as well as its advantages. Although the integrity of the performance measures obtained will be compromised if other network traffic is present, it is more significant that the network need not be devoted to the generator. The

reasons are many. Reloading the system with specialized network software in a coordinated fashion over a moderate-sized network is a monstrous, time-consuming problem. (Sophisticated cross-net operations are not yet a reality.) Alternatively, applications can be run during off-peak times with little degradation of results, or small segments of a network can be temporarily isolated to thoroughly test specified links. This feature basically takes account of the fact that any network, commercial or developmental, is enormously expensive to operate, and it is hardly acceptable to expect all these resources to be devoted to a single task for any length of time.

Another characteristic is the single control point. Although many nodes would be involved in each experiment, it was essential that one person could control it. Any other approach would involve massive communication and coordination difficulties overshadowing any problems in remote control.

One further point must be made before launching into the details. The interactive philosophy was chosen over the static one. What this means is that the experimenter, once he has defined the limits of the experiment, is free to work within them: activating certain paths, retrieving data, enabling tracing, reinitializing statistics, deactivating or activating other paths, and in short, reacting to the results and achieving much more effective use of his and the network's

time.

Module Description

The first module is a stand-alone program that defines the limits mentioned in the preceding paragraph, using the parameters described in section A as well as the tracing option. It is a conversational program that builds the experiment description table (see Appendix B) with one record for each path (generator-receiver pair). Parameters are verified as they are entered to check for proper data type and range. The program has an edit mode in addition to a create mode, and stores the table in the file system, where it can be called up by the experiment control program. Finally, if a file is available describing the network configuration (node characteristics and connections, etc.), then this program should verify that everything described is possible. Some node may not support tracing, or a node name may not exist, and much wailing and gnashing of teeth could be prevented by discovering these mistakes before the experiment. To summarize, any "administrative set-up" associated with the experiment should be siphoned off to an off-line program; to do otherwise would encourage an inefficient use of resources.

All other support programs deal with analyzing the collected data, perhaps in conjunction with previous experimental results. As noted near the beginning of section B, these will

be conveniently but justifiably omitted from this study. A simple program that provides a formatted view of the data is sufficient in lieu of more sophisticated analysis.

The rest of the modules are all part of the on-line experiment environment, and are therefore closely meshed. Figure 1 depicts the control structure at the control node. Figure 2 details the data flow between modules and between nodes.

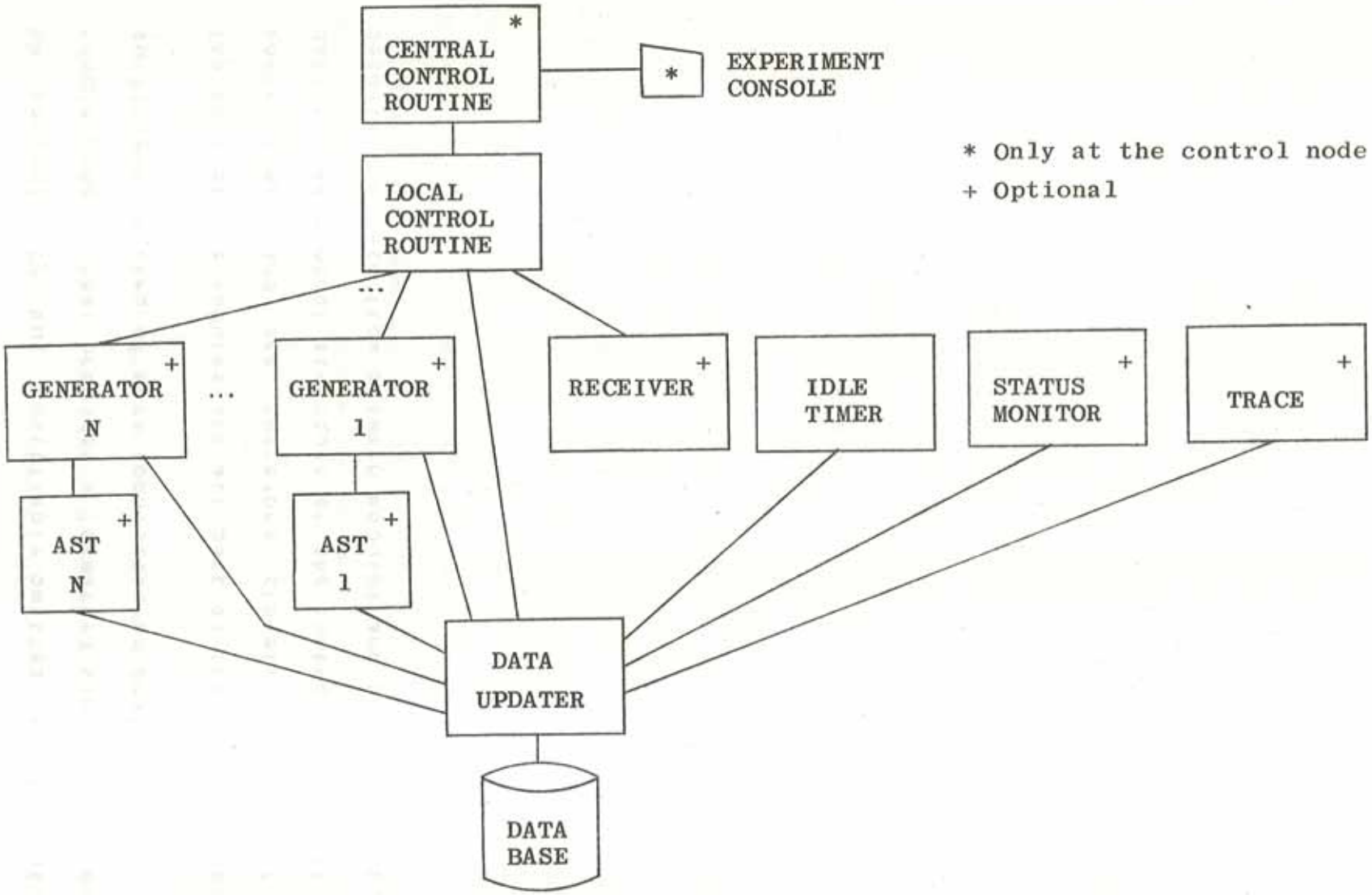

FIGURE 1 CONTROL STRUCTURE AT AN EXPERIMENT NODE

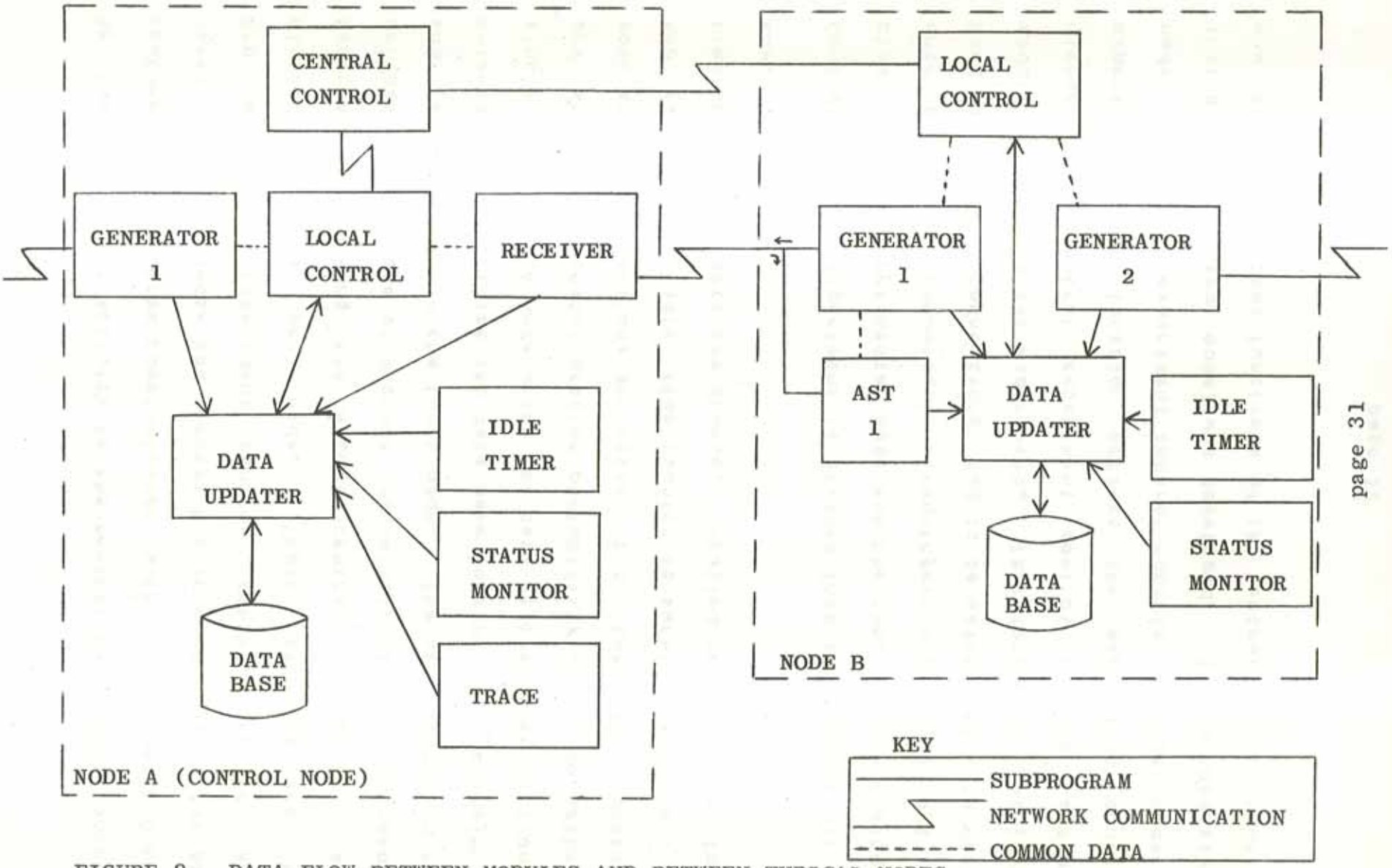

FIGURE 2 DATA FLOW BETWEEN MODULES AND BETWEEN TYPICAL NODES

At the center of activity is the central experiment control routine. This program communicates with the experimenter, parses the commands, and conveys his instructions to the appropriate local experiment control routines which are distributed, one per node. These programs are the lieutenants, obeying the higher central authority, and delegating the chore of message transmission to the generators and receivers within the local node. The generators and receivers in turn utilize the data base updater to maintain a single area in each node where the data described in section B are stored. A monitor program periodically wakes up, gathers some status information, and passes it to the data updater. And finally, an idle time counter leisurely keeps track of time whenever it gets the chance. Detailed descriptions follow.

Control of the experiment is divided into two levels. It is clear that the experimenter must run the show through a single entry point, yet communications requirements would get out of hand if direct connections had to be maintained with every module involved in the experiment. Therefore the two-level hierarchy of central experiment control routine (one per experiment -- its location defines the experiment control node) and local experiment control routines (one per node -- including the control node) was developed. It is necessary now to discuss the functions of the central control routine

before the other modules can be brought into focus.

Central control is the experimenter's gateway to the traffic generator system. Its primary functions are to initialize and carry out the experiment, all under operator instructions. (I will switch to the less formal name of "operator" now that it will be used much more frequently.) Initialization is a far from trivial process, and is automated almost completely to avoid human error. A lot of information must be assembled and then sent off to the proper places if any coordination is to be achieved. The very first step is to retrieve, at the operator's request, the experiment description table that was prepared previously. This table must be broken down into experiment control tables, one for each node. Each of these tables will contain all the information needed for a local control routine to direct the traffic generator functions at its node; it is essentially a partitioning of the experiment description table by nodes.

The first actual use of the network takes place when the central control establishes communication with the local controls. The next act is critical for networks without a common time base: all the nodes must be synchronized or delay measurements, trace data, etc. will be considerably less useful. The method proposed for achieving this makes few assumptions, and is simple to implement. Provided that the level of other network traffic is low, the synchronization is

accurate at least to 1/60 second, the resolution of the test computers' clocks. Central control will play the generator--receiver game with each local control, one at a time. Short messages will be exchanged with a local time stamp on each. From these values, the average one-way delay in each direction can be calculated and, assuming that these delays must be equal, half their difference must be the correction to be applied to the "local time". (Control-node time is arbitrarily made the standard.) This local adjustment is stored at each local node and is made to all time values that are recorded there.

Now the weight of the initialization is thrown onto the shoulders of the local control programs. Each receives its respective experiment control table and sets its house in order accordingly. Generator and receiver modules are prepared for execution, the data updater is started and instructed to initialize its data base, the monitor and idle time counter commence operations, and trace capability is set if requested. Any deviations from intended operation must of course be reported to the control node, as well as reassuring "AOK" messages if all went well. At last the operator is ready to experiment.

Certain commands are necessary if several experiments are to be made in succession without shutting down the entire traffic generator system between runs. Other steps are required to

coordinate the collection of data from several nodes. And some commands are needed just to make life simpler for the operator. A core set of these commands will be sketched here. Details on those that have been implemented can be found in the appendices.

To change the environment of the experiment, the operator may activate or deactivate specific links. These correspond to the records of the experiment description table. Tracing at specific nodes can also be enabled or disabled. The characteristics of each link (such as message length) should not be allowed to change, however. This destroys the integrity of the experiment description table and allows too much freedom to trifle with details. Both encourage poor experiment design. If it is desirable to have links of slightly varying characteristics, then several entries can be made at experiment description time, and they can be activated individually. It may seem somewhat disdainful to make this restriction, but discipline must be instilled in a tool's user as well as in its developer.

An operator must also be allowed to retrieve results and reinitialize data bases in order to make more than one experiment. These two functions reintroduce the need for synchronization. For if data from different nodes are not initialized at the same time or collected at the same time, then they really span different time periods and do not accu-

rately reflect the same set of events. Again, central control will make it perform properly. When it made its first synchronization, maximum outgoing delays to each node were noted. These are now used to calculate a time stamp which is post-dated by several times the maximum delay in the network. This will not amount to more than a few seconds for a small network. When each local control receives one of these instructions, it waits until the designated moment to flip a bit that enables/disables the data updater. Thus all data bases are reinitialized or retrieved simultaneously. If due to extremely unfortunate circumstances one or more of the commands arrives late, then the operator is notified and may try again.

Other convenient commands might involve running an experiment from a command file or invoking simple analyzers for on-line aid in formulating the next experiment. There is no end to the elaborations one can make in this area. Needless to say, that will not be done here.

The parameters that local control received from central control are stored in an area shared with the generator and receiver programs. Each node will support several generators but only one reentrant copy is necessary; this saves space and the parameter area goes a long way to helping achieve reentrancy. The generator is run when the local control routine activates it at the operator's request. It halts when it

sees its abort flag set after a deactivate command. This cooperative abort provides a much cleaner and quicker interface than an executive abort which would pull the generator out of memory kicking and screaming. In between the activation and deactivation is where the interesting things happen. The generator uses all the parameters of section A to construct messages and to time their departures. The only restriction on size is that every message must be long enough to carry its time stamps. After each connection is established, or each message sent, or each reply received, the generator calls the data updater program with the appropriate parameters to keep the data base current.

Under typical traffic conditions, the generator will spend most of its time dormant. Once it has determined when the next message is to be sent, it waits on a clock for the necessary amount of time. It is awakened just in time to send the message. Another matter worth mentioning is the reception of return messages. Since delay measurements are a primary goal of this project, it is absolutely required that the returning messages be intercepted precisely when they are received. This necessitates an asynchronous task that is activated by the generator each time a two-way message is sent and which shares the generator's parameters. The Asynchronous System Trap mechanism in RSX-11M satisfies this requirement and has been used here to properly set up a simple program

which waits for a message, calls the data updater and terminates. This module is called AST in figures 1 and 2.

The receiver module is conceptually much simpler than a generator. Since the only decision it is ever required to make is whether or not to send a message back (again, instructed by local control), there need be only one receiver (logically, as well as physically) per node. This program simply sits around waiting for a message on any active connection. Then it calls the data updater and possibly adds its own time stamp to the message and sends it whence it came.

The monitor module is oblivious to all traffic passing through the node. Its sole function is to periodically read the network status tables via commands or programmatic interfaces and extract error counts, queue lengths, etc. as outlined in section B. It passes this information to the data updater. Since this data does not accumulate as quickly as individual statistics on messages, it is relatively safe to include it in raw form (not summarized, as the other statistics). A threshold must be maintained, however, and if the allotted space becomes filled with this type of information, the data updater will return with a status indicating that the monitor should shut itself off. It can be reactivated the next time that the node's data base is initialized.

Finally we come to the much-mentioned data updater. This

program is essentially everybody's subroutine for write access to the node's data base. There are several reasons for this strategy. Mainly, a shared data base is easiest to initialize, retrieve and control. A simple instruction from local control is sufficient to prevent or allow further updates. A fortunate side-effect is the ease with which changes to the data base format can be accomplished (particulary useful during development!). The data base contains data on message statistics, monitor statistics, and tracing. The latter two require threshold mechanisms as outlined above. This also is the responsibility of the data updater.

The last module can be quickly summarized. The low-priority idle time counter loops a number of times that has been calculated to require a known time period and then calls the data updater (of course) to increment a counter. Subtracting this time from the duration of the experiment yields the processor time consumed by the experiment.

It would be fitting at this point to reiterate the utility of the overall control structure by looking at it from another angle. All intra-node communications take place via common data areas. All inter-node communications, with the exception of generated messages, take place only between the central and respective local experiment control routines. This latter restriction eliminates potential chaos by establishing a strict but natural hierarchy for control and data transfer.

IV. Implementation and Recommendations

A design for a general-purpose network traffic generator has been presented. It emphasizes ease in experiment preparation and flexibility in control, as well as useful statistics collection. It is now time to discuss an implementation of this design. This discussion will focus on the functions performed by the implementation, difficulties encountered, recommendations for simple enhancements and added features, and recommendations for its efficient use.

A. Summary

The traffic generator system that was actually implemented can be most briefly summarized by pointing out that it contains every major feature described in the preceding chapter with the exceptions of tracing and status monitoring. It was written for DECnet under the RSX-11M operating system in the summer and fall of 1977. At that time, the network software was not released and was still undergoing development; consequently, any unilateral tinkering with it would have had extremely short-lived support. This is the main reason why

tracing and monitoring were forsaken in this implementation. Of all the functions described, only these must be able to examine the internal network data structures. More will be said on this later.

The following programs were written:

experiment description file formatter and editor
central experiment control routine
local experiment control routine
traffic generator
traffic receiver
data updater
idle timer

The first program essentially leads one by the hand in developing an experiment description file. For each path description, each parameter is asked for by name and validated as it is entered. The entire file can be displayed, and each field can be individually edited by specifying the path number and the field name. The fields are all described in Appendix B, including some that were left out of chapter III for clarity. For example, this program assigns numbers to generators on a per node basis for administrative purposes, but the user will never need to see these numbers, let alone be requested to define them.

The central control routine performs all its designed functions. It reads the experiment description file, partitions it into experiment control tables (also described in Appendix B) for each node, synchronizes each local control

routine, and executes the operator's commands. The implemented commands are: activate specified paths, deactivate specified paths, exit from the experiment, initialize specified data bases, and retrieve specified data bases. When performance data is retrieved, the central control routine performs an integrity check to ensure that the environment of the experiment was stable while each retrieved data base was active. This is accomplished by incrementing a centrally-stored series number whenever a command is issued to alter the traffic pattern. The local series number is updated only when the local data base is reinitialized. If these two numbers match, then the data is stored away in a user-specified file, along with a field describing the duration of this experiment series.

The local control routine carries out all instructions received from the central routine. No other program communicates with the central control routine. The other local control duties are initialization, which is done top-down in the control hierarchy, and experiment termination, which is performed bottom-up.

Generators and receivers were built to perform all the designed functions except for the ability to add a special header flag marking a message for tracing. Messages can be sent one-way or round trip, may wait for acknowledgement or not, can vary in length, etc.

The data updater updates the data base on request from the receiver, the generators, or the idle timer. It also formats the data for the local control routine during retrieval. However, it need not bother this time with monitor status or trace data. The data base is detailed in Appendix B.

The idle timer was tested as the sole running application program to determine the number of loops it must execute to consume one second of CPU time. This value was incorporated into the final version, and the timer calls the data updater after every accumulated idle second to increment a counter.

All coding was done in Fortran, except for a single subroutine to set up the asynchronous trap routine to capture a returning message as it arrived from the traffic receiver. The Macro-11 assembly language was necessary to accomplish that, while the use of Fortran speeded up the bulk of the program preparation.

B. Some Difficulties Encountered

Other than the problems encountered with initial unfamiliarity with the operating system details, there were few serious problems that had to be overcome. Most problems that did crop up had to do with the use of Fortran. Unfortunately, the system that was available was used primarily for assembly program development and therefore was not configured for Fortran use. Needless to say, experience indicates that this

circumstance should be avoided or at least properly adjusted as early as possible. Problems persisted after Fortran was installed, but their severity decreased exponentially over time. Another difficulty has been previously explained. The need for an asynchronous receiver of returned messages, which could communicate with the generator, had to be resolved by using features found only in the assembly language. The linkage of the compiled and assembled code was easily accomplished by following the Fortran control and parameter passing conventions.

The greatest disadvantage of the use of the available Fortran installation was its relatively inefficient use of memory. This was apparent both in the greater amount of space consumed by equivalent code, and in the extra burden of the language-dependent processors. To come to the point, these forces conspired to devour memory so that, quite early in testing, overlays became necessary. Keep in mind that the generator system was the only application running at the time. (To be fair to the computer, it should be noted that the poor machine was without memory management.) Fortunately this dilemma occured only at the control node which is distinguished by the presence of the central control routine. This is by far the largest program. The solution turned out to be trivial since the design of both the central and local control routines could each be neatly packaged into an initialization section

and an interactive experiment control section. These run not only independently, but also serially, so overlay overhead is practically nonexistent.

One further problem was the cause of much consternation. The immediate reconnection of a path after its disconnection would sporadically cause the generator to quit because the previous connection was "still in effect". Introducing a significant delay, such as printing a message after a disconnection, eliminated the enigma, but at the cost of annoying extra output and a far from elegant generator. The disconnect software was naturally highly suspect. This suspicion was reinforced by the repeated failures of every reasonable effort (and quite a few unreasonable ones) to modify the generator and receiver on the assumption that they were the culprits. The inelegant solution remains, although the system bug has presumably been corrected. An alternate solution would involve building another disconnect mechanism, independent of the network disconnect. This basically assumes greater synchronization, whereby the generator would send a special message telling the receiver to "disconnect when finished with all communications." Since the generator will not send this message until its communications are complete there can be no problem of timing.

C. Enhancement

It should be clear by this point that much can be done to make the traffic generator a more powerful tool and a more comfortable one to use.

Several features would be quite simple to implement. These include expanding the command set, decoding error messages, and rewriting the whole thing in assembly language. The latter can be called simple, even though it is quite time-consuming.

The present command set is a bare minimum necessary for running a series of experiments. A help command to summarize the commands and their syntax would be useful to the novice. An atfile command to transfer the input stream from a terminal to a file would relieve the operator of the burden of dealing with the details of repetitive experiments. It would also help eliminate errors. With such a command file, special commands would be needed to specify a delay between commands and to transfer control back to the operator. Lastly, a display_measurement command would be useful to allow the operator to examine experiment results. This would permit interactive experiment design, and need be no more complicated than the simple offline program that currently dumps results onto the terminal.

Another convenience would be clarification of error messages. As a matter of expediency, many error statuses are reported as numbers or in some cryptic abbreviation. As the generator matures, this will no longer be acceptable. Error texts can be stored in a file separate from the central control routine.

Of the straightforward enhancements, the one with the greatest impact is rewriting the entire generator system in assembly language. This would make the object modules smaller, and the code faster. It would eliminate the Fortran network interface and the other language-dependent overhead. In short, it would greatly increase the performance of this performance tool.

The improvements mentioned thus far deal with convenience and speed. Implementation of the following more sophisticated features is aimed more at increasing the generator's utility. The need for some of these is obvious. Others would be useful in special cases.

The status monitor and tracing have been discussed at length in chapter III. The addition of these two features would make a fairly complete traffic generator. One aspect that deserves more mention, however, is dynamic tracing. Since one may be interested in messages that are transmitted in error, a dynamic trace option would be handy. (Omigosh - another parameter!) This would selectively enable tracing for the messages that encounter specified error conditions. Note that

care should be taken in its design because this option can become very complicated to use. It is possible to bring it to the lowest protocol levels to select error conditions particular to the line protocols.

Incidentally, the considerable effort expended on development of the trace capability does not have to be for the exclusive benefit of the performance experiments. It can and should be used as part of a full-time monitoring facility, and switched on or off as needed. Indeed, in the long run, it can be integrated into a self-diagnosing network.

This is a good place to be reminded that the whole area of data analysis has not been addressed in this paper. This might be called the other half of statistics measurement and cannot be wished away in a full implementation.

Another desirable feature is remote operation of the central control routine. Because it is a large program, it may not be feasable for the experimenter's local node to support it. It seems logical then to invoke the principle that brought networks into being in the first place, and bring the power of the larger computer to the user without bringing the whole computer. The virtual control node would only need a small program to pass commands and responses verbatim to the actual control program at the powerful remote node.

Remote operation is also needed for setting up the programs in individual nodes before the experiment can begin. Under DECnet-11M, when a connection is made to a correspondent program, that program must have been in an "installed" state, ready to run. This poses no problem if the network is all in one room (as was the case during testing) or if every node is attended by an operator. But otherwise, there is no one to type all the commands. A general-purpose facility similar to down-line loading would fill this need admirably.

The last enhancement concerns diagnosis of the generator itself. Many programs are scattered over many nodes, and the failure of any one of them has an impact on the experiment. But unless the failure occurs in the experimenter's node and the system operator's terminal is nearby, the failure will typically go undetected until the experiment is terminated. Each local control routine should have the ability to confirm the functioning of its local group of modules, and to inform the central control routine of any abnormalities. This can be most naturally done after each command.

D. Usage

Finally, let us discuss under what circumstances the traffic generator should be used, and how it might be used in the future. For an example of how to use the present implementation, see Appendix A.

First of all, the generator system is not limited to any type of configuration. If a node name and a program name are sufficient to get a message to its destination, then the generator will function. This is really another advantage of writing the package as an application. As long as the interface remains constant, then it does not matter whether the network is local, global, star, ring, hierarchical, or what have you. Only the tracing and status monitoring modules have to be specialized.

Although the system has been tested only on a two-node network, it is capable of being used in its present form on a network of any size or shape. Each node can operate as many generators as it can support, to a maximum of eight. There are no other logical constraints.

The types of experiments that can be performed are quite varied because of the general nature of the generator. They basically fall into two categories: performance measurement and fault diagnosis.

Performance measurement comes into play as soon as an integrated network can be tested, and stays with it through the rest of its life cycle. In the beginning, activity with the generator will be confined to benchmarks and verification of message transfer. These are mainly measurements of a gross level. As the network matures and its insides stabilize, the

programmatic interfaces to the network software can be implemented. This window will allow more detailed statistics about states and events. As the network becomes more of a tool and less of a development project, the generator will evolve into a monitoring device. Some functions will be used sporadically, such as loading a new line to measure its capacity in a new environment. Other features will be used periodically to monitor traffic levels, memory usage, congestion trends, etc.

This naturally leads into the second category, fault diagnosis. Ideally, appropriate tests will be automatically initiated when key indicators approach dangerous levels, or when a component failure is reported. The resulting measurements will then be used to trigger perhaps even more sophisticated routines, or some super-operator will act on them. The generator system would clearly play only a small, although important, part of such a system. Less ideally, failures, threshold violations, and user complaints will be reported to the nearest operator or to some central authority. From there, a test will be performed to verify that two nodes are connected or to check that a certain line remains capable of operating near its stated capacity. In this way, both hard and soft failures may be diagnosed, and one can get better clues as to which component may be at fault.

V. References

A. Outside References

1. "A Computer Network Monitoring System", David E. Morgan, et al., IEEE Transactions on Software Engineering, September 1975.

2. "The Network Control Center Program", Bolt Beranek and Newman, Inc., February 1973, distributed by NTIS, update edition of September 1974.

3. "The Network Control Center for the APRA Network", A. A. McKenzie, et al., IEEE 1972 International Conf. on Computer Communication, pp. 185-191.

4. "The ARPA Network Control Center", Alexander A. McKenzie, Bolt Beranek and Newman, Inc., Fourth Data Communication Symposium, October 1975.

5. "ARPANET: Design, Operation, Management, and Performance", Network Analysis Corp., April 1973.

6. "Performance Criteria for Digital Data Networks", U.S. Department of Commerce, Office of Telecommunications, January 1975, distributed by NTIS.

7. "Dynamic Management of Data Networks", Marino F. Saksida, Telecommunications, vol. 11, no. 2, February 1977, pp. 39-42.

8. Queueing Systems, Vol. 2: Computer Applications, Leonard Kleinrock, John Wiley and Sons, 1976.

9. "The General Electric Network Monitoring System", George H. Wedberg and Louis W. Hauschild, Proceedings of IFIP Congress 74, pp. 24-28

10. "The Network Measurement Machine - A Data Collection Device for Measuring the Performance and Utilization of Computer Networks", NBS Technical Note 912, U.S. Government Printing Office, 1976.

B. DEC Project Descriptions and Reports

11. "Digital Equipment Corporation Network Proposal and Project Plan, Final Report", Network Analysis Corp., April 1977.

12. "Preliminary Project Plan for DECNET/8 Performance Evaluation, J. Gannon, August 1975.

13. "DECNET Performance Evaluation Task Force Final Report", Nicholas Johnson, June 1977.

14. "Preliminary Tool Functional Specifications for DECNET Performance Evaluation", Peter F. Stokely and David M. Johnson, August 1977.

C. DEC Manuals and Brochures

1. "Introduction to Minicomputer Networks", 1974.

2. "DECNET", 1976 (Overview and Technical Summary).

3. "PDP11/34 Processor Handbook", 1976.

4. "RSX-11 DECNET-11 Programmer's Guide and Reference Manual", 1976.

5. "Specification for: RSX-11 Fortran User Intertask Communication Interface to DECNET, version 2", 1977.

6. A multitude of RSX-11M manuals.

APPENDIX A

SAMPLE EXPERIMENT

To help concretize the preceeding discussion, an annotated example would be extremely helpful.

For this appendix, one of the simplest possible examples was chosen. A single logical link is established between a generator and receiver at the same node. All parameters are non-probabilistic and the generator waits for a reply from the receiver before sending another message. The configuration is similar to node A in figure 2 of chapter III, except that the generator and receiver are communicating with each other.

The example will be easiest to describe in chronological order. Therefore the experiment output will be presented first, and then it will be discussed point by point. Please note that user input is underlined.

```
1
   >RUN_CENCIL
2
   EXPERIMENT DESCRIPTION FILE:IESI1.GEN
```

```
3
   NEXPER=      1      NSERIE=     42      NUMBER LINKS=    1
   TGNODE= TELA
   LENAVE=    100.         LENEXP=    0.00
   TIMAVE=   1.00          TIMEXP=    0.00
   TIMBAS= R
   TGFUNC= R
   CONAVE=      4.         CONEXP=    0.00


4
     1     1        1    0    0    0    0    0    0    0
           1        1    0    0    0    0    0    0    0
   IF ALL IS OK, TYPE 'Y' TO CONTINUE:Y
5
   LOCCTL: NEXPER=  1     NSERIE=  42     NUMGEN= 1     NUMMIR= 1
   LOCCTL: EXPERIMENT CONTROL TABLE RECEIVED.
   CENCTL: ALL EXPERIMENT CONTROL TABLES SENT.
6
   CENCTL: NODE 1: DELAY=    0.0241     CLOCK CORR=    0.0098 SECONDS.
   LOCCTL: CORRECTED LOCAL TIME = LOCAL TIME +     0.009766 SECONDS.
   CENCTL: ALL CLOCKS SYNCHRONIZED.
7
   GEN>AC_ALL
   LOCCTL:    0 FAILURES ON INITIALIZING LINKS    1
   OK.
8
   GEN>IN_IELA
   LOCCTL: INITIALIZED.
   OK.
9
   GEN>REI_032978.2_IELA
   TTO  --  ERROR 29
   NO SUCH FILE
   FCS -26, 0 032978 4
     IN    "CEN2  " at 271
     FROM ".MAIN." at 4

   PLEASE IGNORE THE PREVIOUS FCS MESSAGE.
   CENCTL: RETRIEVED NODE TELA.
   LOCCTL: DATA AREA RETRIEVED.
   LOCCTL: INITIALIZED.
10
   GEN>EX
   LOCCTL: EXITING

   CENCTL: EXITING.


11
   >RUN_REIREV
```

```
12
   ENTER FILENAME: 032978.2

13
   TIME OF EXPERIMENT =  3/29/78 16:40:01
   NEXPER=    1     NSERIE=  44
   DURATION OF EXPERIMENT =      50.90  SECONDS.
    1 ACTIVE LINKS:   1
   1 RETRIEVED NODES:   TELA


14
     TELA

   MSGLEN  SIGML SIG2ML MSGTIM  SIGMT SIG2MT   NCON  SIGCM
       69 7.E+03 7.E+05     69 1.E+00 2.E-02     17 7.E+01

   SIG2CM NDISCO NRNDEL  SIGRD SIG2RD  RDMIN  RDMAX
   3.E+02     17     69 1.E+01 3.E+00 1.E-01 4.E-01

   NDELAY   SIGD  SIG2D DELMIN DELMAX
       70 7.E+00 1.E+00 2.E-02 2.E-01


   NARTIM  SIGAT SIG2AT  ATMIN ATMAX  TIMER
       70 1.E+01 2.E+00 8.E-02 2.E-01 2.E+03

   >
```

1: The greater-than sign is the system prompt character. The experiment is initiated with the RUN CENCTL command which starts the central control routine.

2: Central control immediately asks for the name of the experiment description file which is TEST1.GEN.

3: Central control prints the contents of the experiment description file. The first line summarizes the experiment with its main number, series number (see Appendix B) and the maxi-

mum number or generator-receiver pairs. To speed implementation, the constraint was imposed that all generators be in the control node; therefore the source node is not specified here. In this case, it is the same as the target node, TELA.

The message lengths (in bytes) are characterized by their average length and exponential parameter: 100 and 0 respectively. Recall that a zero exp means a constant value. The intermessage time and messages per connection are similarly defined at 1 second and 4 messages/connect. The time base is "wait for reply" and the target task (receiver) function is to return messages. The last six lines would be repeated for each additional link to be tested.

4: Central control presents an extremely concise tabular summary of the experiment. The first column represents the node number, in the order encountered in the experiment description file. The second column shows the number of generator links and receiver links to this node. The rest of the table gives the link numbers associated with this node. For example, if the sixth experiment node generates traffic on the sixth, seventh and eighth links, and receives traffic on the second and eighth links, then its part of the table would look like:

```
6   3     6   7   8   0   0   0   0   0
      2     2   8   0   0   0   0   0   0
```

Finally, central control asks for confirmation on the experiment identification before proceeding.

5: All messages from local control will appear on a terminal at the local node. Since in this case the local and central nodes are identical, messages will be interspersed, showing the interplay of the two programs.

Local control begins by printing the experiment number, series number, and the number of generator and receiver links. This is the local version of the first line of (3) and the second column of (4). Local control then declares that it has received its control table, while central control similarly declares that it has successfully sent all such tables (only one in this case).

6: After synchronization, each control routine reports on message delay and clock differential. The differential here must be zero because both routines use the same clock. Happily, the value reported is below one time unit, which for this computer is 1/60 second.

7: Finally we arrive at the first command. All links are to be activated. Local control reports that it activated link 1 without error. Central control says 'OK', meaning that the command was performed.

8: Since the state of the network was just changed by acti-

vating a link, we must reinitialize any node whose data base will be retrieved. Otherwise that node's data will not be consistent with a single traffic pattern, and the node's series number will not match the central number when the data is retrieved. (See the discussion of series_number on page 63.) So the next command is INitialize node TELA. Again, no problems are reported.

This last step could be automated if every node is initialized after de/activating links.

9: After a while, it is time to collect the data. The retrieve command specifies a filename, 032978.2, and the nodes to be retrieved, TELA. To prevent file overwriting, the file's existence is checked first. Unfortunately, this results in an undesirable file system message, but that can be ignored. Note that local control also automatically does an initialize after a retrieve since that would normally have to be done anyway.

To perform another experiment now, one would de/activate appropriate links, initialize all nodes whose data will be retrieved, and then after a while retrieve the data bases.

10: No more experiments will be performed here, so we EXit. Link deactivation is automatically performed.

11: A simple file dump program called RETREV will exhibit the

data just collected.

12: The filename is the one given to the retrieve command.

13: Administrative information is presented first. The experiment number and the list of active links are sufficient to define the experiment.

14: At last we see the data itself. Since this project was not involved in data analysis, the data is presented in very crude form, just as it was collected.

Data field names beginning with SIG and SIG2 are sums and sums of squares of the preceeding count field. For example, the number of lengths measured was 69, and the sum of message lengths was 7000, and the sum of squares was 700 000. All messages were 100 bytes in length so this is in agreement with the experiment description file. Data field names with MIN and MAX are minimum and maximum values of the preceeding measures.

The first two lines are generator measures concerning message lengths, intermessage times, connections and round trip delays. The third line concerns the one-way delay to the receiver, and is measured there. These three lines are repeated for each generator at the node. The last line concerns the interarrival times of messages at the receiver in this node. The idle time count, TIMER, is also included on this line.

Two thousand idle clock ticks, or about thirty-three idle seconds elapsed at node TELA during this experiment. The idle time was the time spent in the generator waiting for the right moment to send the next message.

APPENDIX B

TABLE DESCRIPTIONS

This section will attempt to give a concise recapitulation of Chapter III, but in a rather more prosaic manner. It is noteworthy that the tables described here are useful in exactly the form presented. Experiment description, node control, and statistics gathering will each require one of the following data structures.

Experiment Description Table

The experiment description table is built off-line and stored in the file system at the experiment control node. It describes the traffic paths that may be activated during an experiment and which nodes may trace messages. The central experiment control routine reads this file as its first action and creates the experiment control tables from it. Each field is briefly described below. Those that were not mentioned in section A are purely administrative in nature and are assigned either by the routine that builds the table or during the experiment.

PER TABLE

experiment_number: for cataloguing

experiment_description: free format field

monitor_interval: used by the status monitor routine as the sampling rate for certain system parameters such as error counts, up/down status, and queue lengths

series_number: identifies the number of the sub-experiment. This is the only field that is changed when the experiment description table is put away after the experiment. It is incremented after each de/activate paths command and after each initialize/retrieve data bases command. The local copy of this number is updated only when the local data base is initialized. When the operator retrieves a data base, the central series number is compared with the local series number. If they do not match, then the operator knows that the experiment environment was changed since this data base was last initialized, in which case the utility of that data is questionable.

PER PATH

path_number: an index to the table's path entries that uniquely identifies each generator-receiver pair

source_node: the node of the generator

source_task: the generator's task name (GEN1, GEN2, ..., GEN8, assigned sequentially for each node). This identifies which copy of the generator is used for this path and which section of the data base is used for that generator.

target_node: the node of the receiver task. Since there is only one receiver per node, no task name is needed.

message_length_mean_value: number of bytes

message_length_exponential_parameter: with the preceding, this identifies the exponential probability distribution of the message lengths. If it is zero, then the length is fixed.

inter_message_time_mean_value: number of clock ticks

inter_message_time_exponential_parameter: exponential probability distribution for time between sending messages

inter_message_time_base: inter-messge time may be based on either the time the previous message was acknowledged, of the time the reply from the previous message was received

connect_period_mean_value: average number of messages sent during each established path connection

connect_period_exponential_parameter: geometric probability density for the connect period. Since the connect period is measured in number of messages, it cannot assume non-integral values. Connections are immediately reestablished after disconnection.

target_function: describes whether or not the receiver task is to send back the messages it receives. Obviously, if the time_base assumes a reply, then this parameter is already chosen.

trace_flag: signals whether tracing may be enabled

Experiment Control Table

The experiment control table is the local node's subset of the experiment description table. The local control routine receives it from the central control and uses it to control its generators, receivers, status monitor, and data updater. Entries with no descriptions are identical to those in the experiment description table.

experiment_number:

series_number: updated at each initialize command. See the description above for how this is used.

monitor_interval:

trace_flag: informs the local experiment control routine that an extended data base for recording tracing information must be installed

data_update_enable_flag: used during data base initialize and retreive sequences to synchronize statistics recording

PER SOURCE PATH

path_number:

active_status: identifies a path as activated or deactivated

source_data_base_entry: assigned by the central experiment control routine when assembling this table. Each generator has its own area in the data base for recording its information.

target_node:

message_length_mean_value:

message_length_exponential_parameter:

inter_message_time_mean_value:

inter_message_time_exponential_parameter:

inter_message_time_base:

connect_period_mean_value:

connect_period_exponential_parameter:

target_function:

PER TARGET PATH

link_number:

active_status: identifies the path as activated or deactivated

target_function:

Data Base

The data base is divided into three sections. One section absorbs all the data that the status monitor routine generates, another does the same for the tracing data, and the third section contains the statistics gathered by the

generators, the receiver and the idle timer. The first two sections require threshold mechanisms.

The data base itself is accessible only to the data updater and the local control routine. The latter never writes into it and reads from it only to retrieve the data on operator command.

timer: number of ticks (usually 1/60 second) that a low-priority program has been running. This is used to calculate CPU utilization.

MONITOR DATA

node_status: active/inactive status of neighboring nodes

error_counts: errors per link

free_buffer_length: a measure of the congestion in the node

output_queue_length: the backlog of each modem

sent_queue_length: number of messages waiting for acknowledgement

retransmission_queue_length: number of messages waiting for error recovery

conversion_queue_length: number of messages waiting for protocol translation

store_and_forward_queue_length: number of messages in transit waiting for routing

TRACE DATA

message_header: a unique identifier for correlation with trace data from other nodes

arrival_time: time stamp of arrival at the node's network software

output_queue_time: time stamp of entry to output queue (logical transmission)

time_transmitted: time stamp of physical transmission

acknowledge_time: time stamp of aknowledgement or placement on retransmission queue

output_line: number of the output line for routing verification

MESSAGE STATISTICS

interarrival_time: intervals between arrivals at the receiver task. The number of messages, the sum of times, and the sum of squares of the times are recorded, as well as the maximum and minimum times.

N.B. The remaining six categories of data are recorded separately for each link.

message_length: number of messages, sum and sum of squares of message lengths. These three values yield mean value and standard deviation.

inter_message_time: number of times, sum and sum of squares of times

messages_per_connect: number of connects, sum and sum of squares of messages per connect

disconnects: number of disconnects

round_trip_delay: number of round trips, sum and sum of squares of delays, minimum and maximum delays

one_way_delay: number of trips, sum and sum of squares of delays, mimimum and maximum delays

APPENDIX C

COMMAND DESCRIPTIONS

This command set for the traffic generator system emphasizes both flexible experiment control and ease of use. Please note that not all of these commands have been implemented; see Chapter IV. For an example of the use of those that have been implemented, see Appendix A.

Activate <links>

The specified link numbers are looked up in the description table, and the command is routed to the appropriate local control routines. There the generators responsible for those links are activated. Then a return message is sent to central control, and the experimenter is informed.

Atfile <filename>

The specified file is used for command input in the place of the terminal. Commands and messages still appear on the terminal.

Deactivate <links>

This undoes an activate.

Exit

This causes all links to be deactivated, and the experiment to be ended. No data bases are retrieved or saved.

Help

This prints a short description of all the commands.

Initialize <data_bases>

This is meant to be done after the desired links are activated, so that the message generators are in full swing (steady state) when data collection begins. If more than one data base is specified, then they will be initialized synchronously as follows. A post-dated time is included in the initialization messages. Each local control routine immediately initializes the data base and goes to sleep until the specified time, when it re-enables updating.

Measure <arguments>

This will compute and display performance measures from a previously saved data base or bases. Arguments are as

yet undefined, and the only measures to be initially implemented are throughput and delay.

Retrieve <data_bases>

This will cause the specified data bases to be sent to the control node and saved for later processing. Like the initialize command, this command will be executed synchronously, in order that the data in several nodes can be correlated. Otherwise, covering different time spans, the data can only be considered independently. In addition to the data itself, the current state of message generation (i.e., the activated links and their descriptions) will be saved. Also saved will be the last initialize time, and the present time, crucial for throughput calculation. Note that, after changing conditions of the experiment, such as de/activating links, the experimenter is responsible for initializing those data bases to be later retrieved. By failing to initialize, the data bases contain information relating to more than one type of load on the network, and any results will be misleading.

Notrace <nodes | links>

Disables tracing (see Trace)

Terminal

This command at the end of a command file puts the control of the experiment back at the terminal. If the file does not end in either an Exit or a Terminal, Exit is assumed.

Trace <nodes | links> <freq>

Tracing a link causes messages generated on that link to have their trace bit turned on. This is not, however, sufficient to get tracing information about them recorded. Tracing a node causes that node to record information about every trace-enabled message which passes through it. Frequency applies only to tracing links and specifies that every nth message is to be traced.

APPENDIX D

PROGRAM LISTINGS

The following listings are the implementation of the DECnet traffic generator described primarily in Chapter III. Chapter IV details which parts of the design were included and which were not. The previous appendices further clarify the external and internal operation of the generator system.

For ease of use, the program task names in the listings are matched here with the names used throughout this report:

CENCTL - Central Experiment Control Routine

CEN1 - initialization phase

CEN2 - operational phase

LOCCTL - Local Experiment Control Routine

GEN1 - Generator

RMAC - Asynchronous System Trap (AST)

MIRROR - Receiver

DATUP - Data Updater

CPU - Idle Timer (CPU utilization)

KOMMON - System Common Area (Data Base)

```
C      PROGRAM CENCTL - CENTRAL EXPERIMENT CONTROL ROUTINE.
C      WRITTEN BY R. STRAZDAS.
C
C      THIS PROGRAM ASSEMBLES A USER-SPECIFIED EXPERIMENT DESCRIPTION
C      FILE INTO APPROPRIATE EXPERIMENT CONTROL TABLES, AND SENDS THEM
C      OFF TO LOCAL EXPERIMENT CONTROL ROUTINES AT THE RESPECTIVE
C      NODES.
C      IT PERFORMS CLOCK SYNCHRONIZATION
C      AND OTHER SETUP TASKS, INFORMS OF NETWORK STATUS, AND ACCEPTS
C      INTERACTIVE COMMANDS FROM A TERMINAL OR FILE.
C
C      SUBROUTINE CEN1 PERFORMS ALL SETUP TASKS.
C      SUBROUTINE CEN2 ACCEPTS COMMANDS.
C
C
C      DATA DEFINITIONS:
C
C   BAREA  BYTE ARRAY USED FOR SENDING AND RECEIVING NETWORK DATA
C   CLOCK  USED FOR RECORDING THE TIME OF DAY IN A RETRIEVE FILE
C   CMDLIN 80-CHAR COMMAND LINE FROM THE TERMINAL OR @FILE
C   CONAVE GENERATOR MESSAGES/CONNECT MEAN VALUE, AS READ FROM
C          THE EXPERIMENT DESCRIPTION FILE
C   CONEXP GENERATOR MESSAGES/CONNECT GEOMETRIC PROBABILITY PARAMETER,
C          AS READ FROM THE EXPERIMENT DESCRIPTION FILE
C   DELAY  ARRAY OF ONE-WAY TIMES TO EACH NODE, AS DETERMINED DURING
C          SYNCHRONIZATION
C   DELMAX MAXIMUM DELAY TO THE NODES IN A PARTICULAR COMMAND
C   FNAME  USED FOR ACCEPTING AND ASSIGNING FILE NAMES
C   FNODE  USED FOR RECORDING NODE NAMES IN THE RETRIEVE FILE
C   IAREA  ARRAY USED FOR SENDING AND RECEIVING NETWORK DATA
C   IARG   IS 0 WHEN THE CURRENT ARGUMENT OF CMDLIN IS THE LAST
C   IDS    STATUS INDICATOR FOR EXECUTIVE DIRECTIVE CALLS; ALSO USED
C          IN DE/ACTIVATE COMMANDS TO DETERMINE WHETHER A GIVEN NODE
C          IS INVOLVED
C   IOSB_  STATUS BLOCKS FOR NETWORK CALLS
C   L      INDEX FOR CMDLIN; ALSO SOMETIMES A DO LOOP PARAMETER
C   LENAVE GENERATOR MESSAGE LENGTH MEAN VALUE, AS READ FROM THE
C          EXPERIMENT DESCRIPTION FILE
C   LENEXP GENERATOR MESSAGE LENGTH EXPONENTIAL PROBABILITY PARAMETER,
C          AS READ FROM THE EXPERIMENT DESCRIPTION FILE
C   LENTBL LENGTH OF THE EXPERIMENT DESCRIPTION TABLE (NUMBER OF LINKS)
C   LENTMP THE LENGTH IN CHARACTERS OF TEMP
C   LINK   A LINK NUMBER DECODED FROM TEMP
C   LINKN2 USED TO RECORD THE ACTIVE LINKS FOR THE RETRIEVE FILE
C   LINKSW IS 'Y' IF THE LINK IS ACTIVE, ' ' IF INACTIVE, 'L' IF
C          IN LIMBO (STATUS BEING CHANGED)
C   LNKGEN ARRAY OF LINK NUMBERS ASSOCIATED WITH GENERATORS,
C          INDEXED BY NODE
C   LNKMIR ARRAY OF LINK NUMBERS ASSOCIATED WITH MIRROR TASKS,
C          INDEXED BY NODE
C   LNODE  USED BY SUBROUTINE BFMT1 TO BUILD THE DESTINATION
C          DESCRIPTOR AREA
```

```
C   LOC     RETURN MARKER FOR THE INTERNAL SUBROUTINE 'DELAY'
C   LTASK   CHARACTER ARRAY CONSTANT FOR TASKNAME 'LOCCTL'
C   L1__    CHARACTER CONSTANT FOR CHARACTER __
C   L_____  COMMAND NAME CHARACTER ARRAY FOR COMMAND _____
C   MNODE   INDEX FOR NODE NAMES IN RETRIEVE COMMAND
C   MSTAT   STATUS OF THIS TASK'S NETWORK DATA QUEUE
C   NEXPER  EXPERIMENT NUMBER, AS READ FROM THE EXPERIMENT
C           DESCRIPTION FILE
C   NGEN    ARRAY OF NUMBER OF GENERATORS AT EACH NODE
C   NMIR    ARRAY OF NUMBER OF LINKS TO MIRROR TASK AT EACH NODE
C   NODE    ARRAY OF NODE NAMES INVOLVED IN THIS EXPERIMENT
C   NODESW  USED TO KEEP TRACK OF WHICH NODES ARE INVOLVED IN
C           A COMMAND
C   NREC    INDEX FOR RETRIEVE FILE
C   NSERIE  EXPERIMENT SERIES NUMBER, AS READ FROM THE EXPERIMENT
C           DESCRIPTION FILE, AND UPDATED DURING THE EXPERIMENT
C   OUTMSG  BYTE ARRAY FOR EXTRA INFORMATION IN CONNECT MESSAGES
C   RAREA   REAL ARRAY USED FOR SENDING AND RECEIVING NETWORK DATA
C   SIGCL   RUNNING TOTAL OF TIMES FROM CENCTL TO LOCCTL, DURING
C           SYNCHRONIZATION
C   SIGLC   RUNNING TOTAL OF TIMES FROM LOCCTL TO CENCTL, DURING
C           SYNCHRONIZATION
C   SWITCH  IS 'Y' IF IN @FILE MODE; ALSO USED AS STATUS FOR
C           SUBROUTINES BACC AND BFMT1
C   TEMP    CHARACTER ARRAY FOR THE CURRENT ARGUMENT FIELD IN CMDLIN
C   TGFUNC  TARGET FUNCTION (FOR MIRROR ROUTINE), AS READ FROM THE
C           EXPERIMENT DESCRIPTION FILE
C   TGNODE  TARGET NODES, AS READ FROM THE EXPERIMENT DESCRIPTION
C           FILE
C   TGTBLK  ACCESS CONTROL INFORMATION AND DESTINATION DESCRIPTOR
C           AREA BUILT BY SUBROUTINES BACC AND BFMT1
C   TIMAVE  GENERATOR INTERMESSAGE TIME MEAN VALUE, AS READ FROM THE
C           EXPERIMENT DESCRIPTION FILE
C   TIMBAS  TIMING BASE FOR GENERATOR, AS READ FROM THE
C           EXPERIMENT DESCRIPTION FILE
C   TIMEXP  GENERATOR INTERMESSAGE TIME EXPONENTIAL PROBABILITY
C           PARAMETER, AS READ FROM THE EXPERIMENT DESCRIPTION FILE
C   T1      ARRAY OF TIMES USED FOR SYNCHRONIZATION
C   T2      ARRAY OF TIMES USED FOR SYNCHRONIZATION
C
C
C
C      LUN DEVICE
C      --- ------
C        1  NS (NETWORK DATA QUEUE)
C        2  SY
C        3  EXPERIMENT DESCRIPTION FILE
C        4  RETRIEVE FILE
C        5  TI
C        6  CL
C        7  @FILE
C        8  NS (LINK TO LOCCTL AT NODE 1)
```

```
C       9   NS (LINK TO LOCCTL AT NODE 2)
C       :   :
C
C
C
      LOGICAL*1 LINKSW(10), NODE(6,5)
C     NODE TABLES FOR UP TO 5 NODES (NODE NAME, LINK NUMBERS TO
C     GENERATORS, LINK NUMBERS TO MIRRORS)
      COMMON /CEN/ LNKGEN(8,5), NGEN(5), DELAY(5),
     X LINKSW, NODE, NNODE, NEXPER, NSERIE, LENTBL
      CALL CEN1
      CALL CEN2
      END



      SUBROUTINE STUFF
C     SUBPROGRAM TO PUT (CMDLIN(J),J=L,...) INTO BYTE ARRAY TEMP,
C     UNTIL CMDLIN(J) IS BLANK OR A COMMA.  L, THE CMDLIN INDEX,
C     IS UPDATED.
      BYTE CMDLIN(80), TEMP(13), L1COM, L1SP
      COMMON CMDLIN, L, IARG, LENTMP, TEMP
      DATA L1COM, L1SP/',',' '/
      DO 50 J=1,13
50    TEMP(J) = L1SP
      DO 100 J=1,13
      IF (CMDLIN(L+J).EQ.L1COM.OR.CMDLIN(L+J).EQ.L1SP) GO TO 200
100   TEMP(J) = CMDLIN(L+J)
200   L = L + J
      LENTMP = J - 1
      IF (CMDLIN(L).EQ.L1SP) IARG = 0
      RETURN
      END
```

```
C       SUBROUTINE CEN1.
C       THIS PERFORMS THE SETUP TASKS FOR CENCTL.
C
C
C
        SUBROUTINE CEN1
        BYTE TGTBLK(72), LNODE(6), LTASK(6), OUTMSG(8)
        LOGICAL*1 BAREA(200)
        LOGICAL*1 L1Y, SWITCH, L1SP
        LOGICAL*1 LINKSW(10), NODE(6,5)
C       EXPERIMENT DESCRIPTION TABLE (3 LINES)
        LOGICAL*1 TGNODE(6,10), TIMBAS(10), TGFUNC(10)
        REAL LENAVE(10), LENEXP(10)
        DIMENSION TIMAVE(10),TIMEXP(10),CONAVE(10),CONEXP(10)
        DIMENSION FNAME(3), T1(11), T2(10)
        DIMENSION IOSB1(2), IAREA(100), RAREA(50), MSTAT(3)
C       NODE TABLES FOR UP TO 5 NODES (NODE NAME, LINK
C       NUMBERS TO GENERATORS, LINK NUMBERS TO MIRRORS)
        DIMENSION LNKMIR(8,5), NMIR(5)
        COMMON /CEN/ LNKGEN(8,5), NGEN(5), DELAY(5),
       X LINKSW, NODE, NNODE, NEXPER, NSERIE, LENTBL
        EQUIVALENCE (OUTMSG(1), NEXPE),(OUTMSG(3),NSERI),
       X (OUTMSG(5),NUMGEN),(OUTMSG(7),NUMMIR),
       X (BAREA,IAREA), (RAREA,IAREA), (T2,IAREA)
        DATA L1Y,L1SP/'Y',' '/
        DATA NODE/'T','E','L','A',' ',' ', 24*'X'/
        DATA LTASK/'L','O','C','C','T','L'/
C
C       GET THE EXPERIMENT DESCRIPTION FILE.
C
1       TYPE 10
10      FORMAT ('$EXPERIMENT DESCRIPTION FILE:')
        ACCEPT 20, FNAME
20      FORMAT(3A4)
        CALL ASSIGN (3,FNAME,12)
        DEFINE FILE 3 (11,16,U,NXXXXX)
C
C       SET UP THE EXPERIMENT DESCRIPTION TABLE.
C       (10 ENTRIES MAX)
C
C       INITIALIZE THE NUMBER OF GENERATORS AND MIRROR LINKS AT
C       EACH NODE.
        DO 25 I=1,5
        NGEN(I) = 0
25      NMIR(I) = 0
        NNODE = 1
        READ (3'1) NEXPER, NSERIE, LENTBL
        TYPE 26, NEXPER, NSERIE, LENTBL
26      FORMAT (//' NEXPER= ',I5,5X,'NSERIE= ',I5,5X,
       X 'NUMBER LINKS= ',I3)
        IF (LENTBL.EQ.0) GO TO 9999
        NEXPE = NEXPER
```

```
      NSERI = NSERIE
      DO 35 J=1,LENTBL
      READ (3'J+1)(TGNODE(K,J),K=1,6),LENAVE(J),LENEXP(J),
     X TIMAVE(J),TIMEXP(J),TIMBAS(J),TGFUNC(J),CONAVE(J),CONEXP(J)
      LINKSW(J) = L1SP
      TYPE 1160, (TGNODE(K,J)K=1,6),LENAVE(J),LENEXP(J),TIMAVE(J),
     X TIMEXP(J),TIMBAS(J),TGFUNC(J),CONAVE(J),CONEXP(J)
1160  FORMAT (' TGNODE= ',6A1/
     X ' LENAVE= ',F6.0,6X,
     X ' LENEXP= ',F6.2/
     X ' TIMAVE= ',F6.2,6X,
     X ' TIMEXP= ',F6.2/
     X ' TIMBAS= ',A1/
     X ' TGFUNC= ',A1/
     X ' CONAVE= ',F6.0,6X,
     X ' CONEXP= ',F6.2//)
C     CALCULATE NUMBER OF GENERATORS AND MIRROR AT EACH NODE.
      NGEN(1) = NGEN(1) + 1
      LNKGEN(NGEN(1),1) = J
C     CHECK WHETHER THE TARGET NODE HAS BEEN NOTED YET.
      DO 30 I=1,NNODE
      DO 28 K=1,6
      IF (NODE(K,I).NE.TGNODE(K,J)) GO TO 30
28    CONTINUE
C     NODE HAS BEEN NOTED BEFORE
      GO TO 32
30    CONTINUE
C     NEW NODE
      NNODE = NNODE + 1
      DO 31 K=1,6
31    NODE(K,NNODE) = TGNODE(K,J)
32    NMIR(NNODE) = NMIR(NNODE) + 1
      LNKMIR(NMIR(NNODE),NNODE) = J
35    CONTINUE
C     PRINT TABLES TO CHECK THAT IT WORKS.
      DO 36 J=1,NNODE
      TYPE 37,J,NGEN(J),(LNKGEN(K,J),K=1,8),NMIR(J),
     X (LNKMIR(K,J), K=1,8)
37    FORMAT (2X,I2,3X,I2,3X,8(2X,I2)/7X,I2,3X,8(2X,I2))
36    CONTINUE
C     VERIFY THAT THIS IS THE DESIRED EXPERIMENT.
      TYPE 40
40    FORMAT ('$IF ALL IS OK, TYPE ''Y'' TO CONTINUE:')
      ACCEPT 403, SWITCH
403   FORMAT (1A1)
      IF (SWITCH.EQ.L1Y) GO TO 50
      CALL CLOSE(3)
      GO TO 1
50    I = 0
      CALL ASNLUN (1,'NS',0,IDS)
      IF (IDS.NE.1) GO TO 9008
      CALL OPNNTW (1,IOSB1,MSTAT,NNODE)
```

```
      IF (IOSB1(1).NE.1) GO TO 9000
C
C     SEND THE EXPERIMENT CONTROL TABLES TO THE NODES.
C
      DO 100 I=1,NNODE
C     FIRST CREATE THE LOGICAL LINK TO THE NODE.
      CALL BACC (SWITCH,TGTBLK,,,,,,)
      IF (SWITCH.EQ..FALSE.) GO TO 9014
      DO 101 J=1,6
101   LNODE(J) = NODE(J,I)
      CALL BFMT1 (SWITCH,TGTBLK,6,LNODE,,6,LTASK)
      IF (SWITCH.EQ..FALSE.) GO TO 9016
      NUMGEN = NGEN(I)
      NUMMIR = NMIR(I)
      CALL ASNLUN (I+7,'NS',0,IDS)
      IF (IDS.NE.1) GO TO 9008
      CALL CONNTW (I+7,IOSB1,TGTBLK,8,OUTMSG,,)
      IF (IOSB1(1).NE.1) GO TO 9002
C
C     SEND GENERATOR INFORMATION.
C
      IF (NUMGEN.EQ.0) GO TO 120
      DO 110 J=1,NUMGEN
C     LINK NUMBER
      IAREA(1) = LNKGEN(J,I)
C     TARGET NODE, ETC.
      DO 105 K=1,6
105   BAREA(K+2) = TGNODE(K,IAREA(1))
      RAREA(3) = LENAVE(IAREA(1))
      RAREA(4) = LENEXP(IAREA(1))
      RAREA(5) = TIMAVE(IAREA(1))
      RAREA(6) = TIMEXP(IAREA(1))
      RAREA(7) = CONAVE(IAREA(1))
      RAREA(8) = CONEXP(IAREA(1))
      BAREA(33) = TIMBAS(IAREA(1))
      BAREA(34) = TGFUNC(IAREA(1))
      CALL SNDNTW (I+7,IOSB1,34,IAREA)
      IF (IOSB1(1).NE.1) GO TO 9004
110   CONTINUE
C
C     SEND MIRROR INFORMATION
C
120   IF (NUMMIR.EQ.0) GO TO 100
      DO 200 J=1,NUMMIR
C     LINK NUMBER
      IAREA(1) = LNKMIR(J,I)
C     TARGET FUNCTION
      BAREA(3) = TGFUNC(IAREA(1))
      CALL SNDNTW(I+7,IOSB1,3,IAREA)
      IF (IOSB1(1).NE.1) GO TO 9006
200   CONTINUE
100   CONTINUE
```

```
C
C       SYNCHRONIZE CLOCKS.
C
        TYPE 201
201     FORMAT (' CENCTL: ALL EXPERIMENT CONTROL TABLES SENT.')
        DO 300 I=1,NNODE
        T1(1) = SECNDS(0.)
        DO 310 J=1,10
        CALL SNDNTW (I+7,,100,IAREA)
        CALL RECNTW (I+7,,100,IAREA)
310     T1(J+1) = SECNDS(0.)
        SIGCL = 0.
        SIGLC = 0.
C       IN CALCULATING DELAY AND CLOCK CORRECTION, IGNORE THE
C       FIRST TIME VALUES SINCE THE TARGET TASK WAS LIKELY
C       CHECKPOINTED BEFORE THE FIRST SEND.
        DO 320 J=2,10
        SIGCL = SIGCL + T2(J) - T1(J)
320     SIGLC = SIGLC + T1(J+1) - T2(J)
C       DELAY AND CLOCK CORRECTION FOLLOW.
        DELAY(I) = (SIGLC/9. + SIGCL/9.)/2.
        RAREA(1) = DELAY(I) - SIGCL/9.
        TYPE 325, I, DELAY(I), RAREA(I)
325     FORMAT (' CENCTL: NODE',I2,': DELAY=',F10.4,5X,
     X  'CLOCK CORR=',F10.4,' SECONDS.')
        CALL SNDNTW (I+7,,4,IAREA)
C       CHECK WHETHER TASKS CPU AND DATUP WERE INVOKED.
        CALL RECNTW (I+7,,4,IAREA)
        IF (IAREA(1).NE.0) TYPE 301, (NODE(K,I),K=1,6)
301     FORMAT (' CENCTL: TASK CPU AT NODE ',6A1,' COULD NOT',
     X  ' BE INVOKED.')
        IF (IAREA(2).NE.0) TYPE 302, (NODE(K,I),K=1,6)
302     FORMAT (' CENCTL: TASK DATUP AT NODE ',6A1,' COULD ',
     X  'NOT BE INVOKED.')
300     CONTINUE
        TYPE 305
305     FORMAT (' CENCTL: ALL CLOCKS SYNCHRONIZED.')
        RETURN
C
C
C
C       COPY THE LATEST SERIES NUMBER TO THE EXPERIMENT
C       DESCRIPTION FILE, AND EXIT.
9999    WRITE (3'1) NEXPER, NSERIE, LENTBL
        CALL CLOSE (3)
        CALL EXIT
9000    TYPE 9001, IOSB1(1)
9001    FORMAT (' OPNNT UNSUCCESSFUL ON CENCTL. IOSB1(1)= ',I3)
        GO TO 9999
9002    TYPE 9003, IOSB1, LNODE, LTASK
9003    FORMAT (' CONNTW UNSUCCESSFUL ON CENCTL. IOSB1= ',I3,',',
     X  I3/8X,'NODE= ',6A1,5X,'TASK= ',6A1)
```

```
      GO TO 9999
9004  TYPE 9005, I, J, IOSB1(1)
9005  FORMAT (1X,I3,'TH LINK, ',I2,'TH GEN SNDW FAILED ON ',
     X 'CENCTL. IOSB1(1)=',I3)
      GO TO 9999
9006  TYPE 9007, I, IOSB1(1)
9007  FORMAT (1X,I3,'TH LINK MIR SNDW FAILED ON CENCTL.  ',
     X 'IOSB1(1)=',I3)
      GO TO 9999
9008  TYPE 9009, I, IDS
9009  FORMAT (' ASNLUN FAILED ON CENCTL.  I=',I2,5X,'IDS=',I3)
      GO TO 9999
9014  TYPE 9015
9015  FORMAT (' BACC FAILED ON CENCTL.')
      GO TO 9999
9016  TYPE 9017
9017  FORMAT (' BFMT1 FAILED ON CENCTL.')
      GO TO 9999
      END
```

```
C       SUBROUTINE CEN2.
C       THIS PERFORMS THE INTERACTIVE COMMANDS FOR CENCTL.
C
C
        SUBROUTINE CEN2
        LOGICAL*1 CMDLIN(80) TEMP(13)
        LOGICAL*1 L1Y,SWITCH,L1SP,L1A,L1L,NODESW(5),FNODE(6,5)
        LOGICAL*1 LINKSW(10), NODE(6,5)
C       COMMAND NAMES
        LOGICAL*1 LACTIV(9),LATFIL(7),LDEACT(11),LEXIT(5),LHELP(5),
     X LINITI(11),LMEASU(8),LRETRI(9),LNOTRA(8),LTERMI(9),LTRACE(6)
        DIMENSION IOSB1(2), IOSB2(2), IAREA(100), RAREA(50),
     X CLOCK(2), LINKN2(10), FNAME(3)
C       NODE TABLES FOR UP TO 5 NODES (NODE NAME, LINK NUMBERS
C       TO GENERATORS, LINK NUMBERS TO MIRRORS)
        COMMON /CEN/ LNKGEN(8,5), NGEN(5), DELAY(5),
     X LINKSW, NODE, NNODE, NEXPER, NSERIE, LENTBL
        COMMON CMDLIN, L, IARG, LENTMP, TEMP
        EQUIVALENCE (RAREA, IAREA)
        DATA L1Y,L1SP,L1A,L1L/'Y',' ','A','L'/
        DATA LACTIV,LATFIL,LDEACT,LEXIT,LHELP,LINITI,LMEASU,
     X LRETRI, LNOTRA, LTERMI, LTRACE
     X /'A','C','T','I','V','A','T','E',' ',
     X 'A','T','F','I','L','E',' ',
     X 'D','E','A','C','T','I','V','A','T','E',' ',
     X 'E','X','I','T',' ',
     X 'H','E','L','P',' ',
     X 'I','N','I','T','I','A','L','I','Z','E',' ',
     X 'M','E','A','S','U','R','E',' ',
     X 'R','E','T','R','I','E','V','E',' ',
     X 'N','O','T','R','A','C','E',' ',
     X 'T','E','R','M','I','N','A','L',' ',
     X 'T','R','A','C','E',' '/
C
C
C       THE SWITCH IS ON (= 'Y') IF IN ATFILE MODE.
        SWITCH = L1SP
C
C       ACCEPT A COMMAND, VALIDATE IT, AND PERFORM IT.
C
C       EACH COMMAND MUST BE GIVEN WITH TWO OR MORE CHARACTERS
C       FOLLOWED BY A SPACE.  ARGUMENTS ARE DELIMITED BY A COMMA.
C       IF AN ERROR IS DETECTED THEN THE LINE IS RETYPED TO THE
C       POINT WHERE THE ERROR WAS DETECTED.
400     TYPE 401
401     FORMAT ('$GEN>')
        DO 402 I=1,80
402     CMDLIN(1) = L1SP
C       IARG IS ZERO WHEN THERE ARE NO MORE ARGUMENTS.
        IARG = 1
        IF (SWITCH.EQ.L1Y) GO TO 450
        ACCEPT 403, CMDLIN
```

```
403   FORMAT (80A1)
      GO TO 500
450   READ (7,403,END=2298) CMDLIN
      TYPE 403, CMDLIN
500   L = 2
      IF (CMDLIN(1).EQ.LACTIV(1).AND.CMDLIN(2).EQ.LACTIV(2)) GO TO 2000
      IF (CMDLIN(1).EQ.LATFIL(1).AND.CMDLIN(2).EQ.LATFIL(2)) GO TO 2100
      IF (CMDLIN(1).EQ.LDEACT(1).AND.CMDLIN(2).EQ.LDEACT(2)) GO TO 2200
      IF (CMDLIN(1).EQ.LEXIT(1).AND.CMDLIN(2).EQ.LEXIT(2)) GO TO 2300
      IF (CMDLIN(1).EQ.LHELP(1).AND.CMDLIN(2).EQ.LHELP(2)) GO TO 2400
      IF (CMDLIN(1).EQ.LINITI(1).AND.CMDLIN(2).EQ.LINITI(2)) GO TO 2500
      IF (CMDLIN(1).EQ.LMEASU(1).AND.CMDLIN(2).EQ.LMEASU(2)) GO TO 2600
      IF (CMDLIN(1).EQ.LNOTRA(1).AND.CMDLIN(2).EQ.LNOTRA(2)) GO TO 2700
      IF (CMDLIN(1).EQ.LRETRI(1).AND.CMDLIN(2).EQ.LRETRI(2)) GO TO 2800
      IF (CMDLIN(1).EQ.LTERMI(1).AND.CMDLIN(2).EQ.LTERMI(2)) GO TO 2900
      IF (CMDLIN(1).EQ.LTRACE(1).AND.CMDLIN(2).EQ.LTRACE(2)) GO TO 3000
510 TYPE 520
520   FORMAT (' NO SUCH COMMAND.')
C     ERROR IN THE COMMAND LINE
530   TYPE 540, (CMDLIN(J), J=1,L)
540   FORMAT (1X,80A1/)
      GO TO 400
C
C     ACTIVATE COMMAND
C
C     GET THE LINK NUMBERS.  IF THE FIRST ONE IS 'ALL', THEN
C     ACTIVATE ALL LINKS.
C     TO ACTIVATE, SEND A MESSAGE TO THE ASSOCIATED LOCAL CONTROL
C     ROUTINE WITH THE LINK NUMBERS TO ACTIVATE.  IT WILL
C     CONFIRM BY SENDING BACK AN INTEGER STATUS VALUE, AS WELL
C     AS LINK NUMBERS FOR THOSE THAT FAILED TO BE ACTIVATED.
2000  DO 2001 L=3,9
      IF (CMDLIN(L).EQ.L1SP) GO TO 2010
      IF (CMDLIN(L).NE.LACTIV(L)) GO TO 510
2001  CONTINUE
C     THE FIRST INTEGER IN IAREA IDENTIFIES THE COMMAND.
2010  IAREA(1) = 1
C     CODE FROM HERE TO 2096 IS ALSO USED BY THE DEACTIVATE
C     COMMAND
2015  CALL STUFF
      IF (TEMP(1).EQ.L1A.AND.TEMP(2).EQ.L1L.AND.TEMP(3).EQ.L1L
     X .AND.TEMP(4).EQ.L1SP) GO TO 2040
C     FIND WHICH LINKS TO ACTIVATE.
      IF (LENTMP.GT.2) GO TO 2090
      DECODE (LENTMP,2020,TEMP,ERR=2090) LINK
2020  FORMAT (I2)
      IF (LINK.LE.0.OR.LINK.GT.LENTBL) GO TO 2090
      IF (LINKSW(LINK).EQ.L1SP.AND.IAREA(1).EQ.2) GO TO 2029
      IF (LINKSW(LINK).EQ.L1Y.AND.IAREA(1).EQ.1) GO TO 2031
      LINKSW(LINK) = L1L
      GO TO 2035
2029  TYPE 2030, LINK
```

```
2030  FORMAT (' LINK ',I3,' IS ALREADY INACTIVE...')
      GO TO 2035
2031  TYPE 2032, LINK
2032  FORMAT (' LINK ', I3',' IS ALREADY ACTIVE...')
2035  IF (IARG.NE.0) GO TO 2015
      GO TO 2050
C     ACTIVATE ALL LINKS.
2040  DO 2045 J=1,LENTBL
      IF (LINKSW(J).EQ.L1SP.AND.IAREA(1).EQ.1.OR.
     X    LINKSW(J).EQ.L1Y .AND.IAREA(1).EQ.2) LINKSW(J) = L1L
2045  CONTINUE
C     DO THE ACTIVATING.
2050  NSERIE = NSERIE + 1
      DO 2060 J=1,NNODE
C     CHECK IF THE CURRENT NODE HAS ANY GENERATORS.
      IF (NGEN(J).EQ.0) GO TO 2060
      IDS = 0
      DO 2070 K=1,NGEN(J)
      IAREA(K+2) = 0
C     COPY LINK NUMBERS TO BE ACTIVATED.
      IF (LINKSW(LNKGEN(K,J)).NE.L1L) GO TO 2070
      IAREA(K+2) = 1
      IDS = 1
2070  CONTINUE
      IF (IDS.EQ.0) GO TO 2060
      CALL SNDNT (J+7,IOSB1,20,IAREA)
      CALL RECNTW (J+7,IOSB2,20,IAREA)
      IF (IOSB1(1).NE.1) GO TO 2075
      IF (IOSB2(1).NE.1) GO TO 2077
      IF (IAREA(2).EQ.1) GO TO 2058
C     AT LEAST ONE LINK WAS NOT ACTIVATED.
      DO 2072 K=1,NGEN(J)
      IF (IAREA(K+2).GE.0) GO TO 2072
      TYPE 2071, 0 - IAREA(K+2)
C     RESTORE LINK STATUS, SINCE THIS LINK WAS NOT DE/ACTIVATED.
      IF (IAREA(1).EQ.1) LINKSW(LNKGEN(K,J)) = L1SP
      IF (IAREA(1).EQ.2) LINKSW(LNKGEN(K,J)) = L1Y
2071  FORMAT (' CENCTL: LINK ',I3,' WAS NOT DE/ACTIVATED...')
2072  CONTINUE
      GO TO 2058
2074  FORMAT (' CENCTL: COMMAND FAILED AT NODE ',6A1,
     X '.  STATUS= ',I3,5X,'...')
2075  TYPE 2076, (NODE(L,J), L=1,6), IOSB1(1)
2076  FORMAT (' CENCTL: SND TO NODE ',6A1,' FAILED.',
     X '  STATUS= ',I3,5X,'...')
      GO TO 2055
2077  TYPE 2078, (NODE(L,J), L=1,6), IOSB2(1)
2078  FORMAT (' CENCTL: RECW FROM NODE ',6A1,' FAILED.',
     X '  STATUS= ',I3,5X,'...')
C     RESTORE LINK STATUS, SINCE THE COMMAND MAY NOT HAVE
C     BEEN CARRIED OUT.
2055  DO 2056 K=1,NGEN(J)
```

```
      IF (LINKSW(LNKGEN(K,J)).NE.L1L) GO TO 2056
      IF (IAREA(1).EQ.1) LINKSW(LNKGEN(K,J)) = L1SP
      IF (IAREA(1).EQ.2) LINKSW(LNKGEN(K,J)) = L1Y
2056  CONTINUE
      GO TO 2060
C     UPDATE LINK STATUS, SINCE THE COMMAND WORKED AT THIS NODE.
2058  DO 2059 K=1,NGEN(J)
      IF (LINKSW(LNKGEN(K,J)).NE.L1L) GO TO 2059
      IF (IAREA(1).EQ.1) LINKSW(LNKGEN(K,J)) = L1Y
      IF (IAREA(1).EQ.2) LINKSW(LNKGEN(K,J)) = L1SP
2059  CONTINUE
2060  CONTINUE
2079  TYPE 2080
2080  FORMAT (' OK.')
      GO TO 400
2090  TYPE 2095
2095  FORMAT (' BAD LINK NUMBER.')
C     RESTORE LINK STATUS, SINCE THE COMMAND IS INVALID.
      DO 2096 J=1,LENTBL
      IF (LINKSW(J).NE.L1L) GO TO 2096
      IF (IAREA(1).EQ.1) LINKSW(J) = L1SP
      IF (IAREA(1).EQ.2) LINKSW(J) = L1Y
2096  CONTINUE
      GO TO 530
C
C     ATFILE COMMAND
C
2100  DO 2101 L=3,7
      IF (CMDLIN(L).EQ.L1SP) GO TO 2110
      IF (CMDLIN(L).NE.LATFIL(L)) GO TO 510
2101  CONTINUE
2110  CALL STUFF
      TYPE 8888
8888  FORMAT (' [NOT SUPPORTED]')
      GO TO 530
C
C     DEACTIVATE COMMAND
C
C     OPPOSITE OF ACTIVATE COMMAND.  SEE DESCRIPTION THERE.
2200  DO 2201 L=3,11
      IF (CMDLIN(L).EQ.L1SP) GO TO 2210
      IF (CMDLIN(L).NE.LDEACT(L)) GO TO 510
2201  CONTINUE
2210  IAREA(1) = 2
       GO TO 2015
C
2298  TYPE 2299
2299  FORMAT (' [END OF FILE.  EXIT ASSUMED.]')
      GO TO 2310
C
C     EXIT COMMAND
C
```

```
C       TAKES NO ARGUMENTS.  A MESSAGE IS SENT TO ALL NODES,
C       AND A CONFIRMATION STATUS IS RECEIVED.  THE LOCAL
C       CONTROL ROUTINE IS SOLELY RESPONSIBLE FOR SHUTTING
C       OFF GENERATORS, ETC.
2300    DO 2301 L=3,5
        IF (CMDLIN(L).EQ.L1SP) GO TO 2310
        IF (CMDLIN(L).NE.LEXIT(L)) GO TO 510
2301    CONTINUE
2310    IAREA(1) = 3
C       SEND EXIT MESSAGES TO ALL NODES.
        DO 2330 J=1,NNODE
        CALL SNDNT (J+7,IOSB1,2,IAREA)
        CALL RECNTW (J+7,IOSB2,2,IAREA)
        IF (IOSB1(1).NE.1) GO TO 2320
        IF (IOSB2(1).NE.1) GO TO 2325
        IF (IAREA(1).NE.1) TYPE 2074,(NODE(L,J),L=1,6),IAREA(1)
        GO TO 2330
2320    TYPE 2076, (NODE(L,J), L=1,6), IOSB1(1)
        GO TO 2330
2325    TYPE 2078, (NODE(L,J), L=1,6), IOSB2(1)
2330    CONTINUE
        TYPE 2340
2340    FORMAT (/' CENCTL: EXITING.'///)
        NSERIE = NSERIE + 1
        CALL CLSNTW(IOSB1)
        IF (IOSB1(1).NE.1) GO TO 9018
        GO TO 9999
C
C       HELP COMMAND
C
2400    DO 2401 L=3,5
        IF (CMDLIN(L).EQ.L1SP) GO TO 2410
        IF (CMDLIN(L).NE.LHELP(L)) GO TO 510
2401    CONTINUE
2410    TYPE 8888
        GO TO 530
C
C       INITIALIZE COMMAND
C
C       GET THE NODE NAMES.  IF THE FIRST ONE IS 'ALL', THEN
C       INITIALIZE ALL NODES.  COMPUTE THE MAXIMUM DELAY TO THE
C       NODES, USING DELAY VALUES OBTAINED DURING SYNCHRONIZATION.
C       ADD 3 TIMES THIS DELAY TO THE PRESENT TIME, AND SEND THIS
C       TO THE NODES.  THE LOCAL CONTROL ROUTINES WILL INITIALIZE
C       THEIR DATA AREAS AND SYNCHRONOUSLY REENABLE THEIR DATA
C       UPDATING AT THE PRESCRIBED TIME.
2500    DO 2501 L=3,11
        IF (CMDLIN(L).EQ.L1SP) GO TO 2510
        IF (CMDLIN(L).NE.LINITI(L)) GO TO 510
2501    CONTINUE
2510    IAREA(1) = 4
        LOC = 1
```

```
      GO TO 4000
2590  DO 2595 J=1,NNODE
      IF (NODESW(J).NE.L1Y) GO TO 2595
      CALL RECNTW (J+7,IOSB1,2,IAREA)
      IF (IOSB1(1).NE.1) GO TO 2592
      IF (IAREA(1).NE.1) TYPE 2074, (NODE(L,J),L=1,6),IAREA(1)
      GO TO 2595
2592  TYPE 2078, (NODE(L,J), L=1,6)
2595  CONTINUE
C     SAVE THE INITIALIZE TIME.
      T1 = RAREA(2)
      GO TO 2079
C
C     MEASURE COMMAND
C
2600  DO 2601 L=3,8
      IF (CMDLIN(L).EQ.L1SP) GO TO 2610
      IF (CMDLIN(L).NE.LMEASU(L)) GO TO 510
2601  CONTINUE
2610  CALL STUFF
      TYPE 8888
      GO TO 530
C
C     NOTRACE COMMAND
C
2700  DO 2701 L=3,8
      IF (CMDLIN(L).EQ.L1SP) GO TO 2710
      IF (CMDLIN(L).NE.LNOTRA(L)) GO TO 510
2701  CONTINUE
2710  IAREA(1) = 5
      CALL STUFF
      TYPE 8888
      GO TO 530
C
C     RETRIEVE COMMAND
C
2800  DO 2801 L=3,9
      IF (CMDLIN(L).EQ.L1SP) GO TO 2810
      IF (CMDLIN(L).NE.LRETRI(L)) GO TO 510
2801  CONTINUE
2810  IAREA(1) = 6
      CALL STUFF
C     GET THE NAME OF THE FILE IN WHICH THE INFORMATION WILL
C     BE STORED.
      DECODE (12,20,TEMP) FNAME
20    FORMAT (3A4)
      CALL ASSIGN (4,FNAME,12)
      DEFINE FILE 4 (46,38,U,NREC)
      READ (4'1,ERR=2811) I
      GO TO 2890
2811  TYPE 2812
2812  FORMAT (' PLEASE IGNORE THE PREVIOUS FCS MESSAGE.'/)
```

```
      LOC = 2
      GO TO 4000
C     GET THE INFORMATION AND STORE IT.
2820  NREC = 2
      MNODE = 0
      DO 2840 J=1,NNODE
      IF (NODESW(J).NE.L1Y) GO TO 2840
      DO 2830 I=1,9
      CALL RECNTW (J+7,IOSB1,72,IAREA)
      IF (IOSB1(1).NE.1) GO TO 2838
2830  WRITE (4'NREC) (IAREA(K), K=3,36)
C     CHECK THE STATUS AND THE EXPERIMENT DESCRIPTION NUMBER.
      IF (IAREA(1).NE.1) TYPE 2074, (NODE(L,J), L=1,6), IAREA(1)
      IF (IAREA(2).NE.NSERIE) TYPE 2832, (NODE(L,J), L=1,6),
     X IAREA(2), NSERIE
2832  FORMAT (' SERIES NUMBER AT NODE ',6A1,' (',I4,') DOES NOT',
     X ' MATCH THAT IN CENTRAL CONTROL (',I4,').'/
     X ' DATA FROM THIS NODE SHOULD BE USED WITH EXTREME CAUTION.')
C     GET THE NODE NAMES AS THE DATA AREAS ARE RECEIVED.
      MNODE = MNODE + 1
      DO 2835 I=1,6
2835  FNODE(I,MNODE) = NODE(I,J)
      TYPE 2821, (NODE(I,J),I=1,6)
2821  FORMAT (' CENCTL: RETRIEVED NODE ',6A1,'.')
      GO TO 2840
C     RECEIVE ERROR -- THIS NODE WILL NOT BE INCLUDED IN THE FILE.
2838  NREC = NREC - I + 1
      TYPE 2078, (NODE(L,J), L=1,6), IOSB1(1)
      TYPE 2839
2839  FORMAT (' CENCTL: SO THIS NODE IS NOT INCLUDED IN THE FILE.')
2840  CONTINUE
C     RECORD THE ACTIVE LINKS, DATE, TIME, AND ELAPSED TIME
C     SINCE THE LAST INITIALIZE.
      K = 0
      DO 2850 I=1,10
      IF (LINKSW(I).NE.L1Y) GO TO 2850
      K = K + 1
      LINKN2(K) = I
2850  CONTINUE
      CALL IDATE(I,J,L)
      CALL TIME(CLOCK)
      WRITE (4'1) I,J,L,CLOCK,NEXPER,NSERIE,T1,K,(LINKN2(J),J=1,K),
     X MNODE, ((FNODE(I,J), I=1,6), J=1,MNODE)
      CALL CLOSE(4)
      NSERIE = NSERIE + 1
C     NOW INITIALIZE ALL NODES.
      IAREA(1) = 4
      LOC = 1
      GO TO 4070
2890  CALL CLOSE(4)
      TYPE 2895
2895  FORMAT (' CENCTL: THIS FILE IS ALREADY IN USE.')
```

```
      GO TO 530
C
C     TERMINAL COMMAND
C
2900  DO 2901 L=3,9
      IF (CMDLIN(L).EQ.L1SP) GO TO 2910
      IF (CMDLIN(L).NE.LTERMI(L)) GO TO 510
2901  CONTINUE
2910  TYPE 8888
      GO TO 530
C
C     TRACE COMMAND
C
3000  DO 3001 L=3,6
      IF (CMDLIN(L).EQ.L1SP) GO TO 3010
      IF (CMDLIN(L).NE.LTRACE(L)) GO TO 510
3001  CONTINUE
3010  IAREA(1) = 7
      CALL STUFF
      TYPE 8888
      GO TO 530
C
C     INTERNAL SUBROUTINE FOR GETTING NODE NAMES AND
C     CALCULATING MAXIMUM DELAY.
C
4000  DELMAX = 0
      CALL STUFF
C     FIND WHICH NODES TO MANIPULATE.
      DO 4010 I=1,NNODE
4010  NODESW(I) = L1SP
      IF (TEMP(1).EQ.L1A.AND.TEMP(2).EQ.L1L.AND.TEMP(3).EQ.L1L
     X .AND.TEMP(4).EQ.L1SP) GO TO 4070
C     FIRST, CHECK IF THE NODE EXISTS.
4020  DO 4040, J=1,NNODE
      DO 4030 I=1,6
      IF (TEMP(I).NE.NODE(I,J)) GO TO 4040
4030  CONTINUE
      IF (DELMAX.LT.DELAY(J)) DELMAX = DELAY(J)
      NODESW(J) = L1Y
      GO TO 4060
4040  CONTINUE
C     NO MATCH.
      TYPE 4050
4050  FORMAT (' NO SUCH NODE.')
      IF (LOC.EQ.2) CALL CLOSE (4)
      GO TO 530
C     GET ANOTHER ARGUMENT.
4060  IF (IARG.EQ.0) GO TO 4090
      CALL STUFF
      GO TO 4020
C     ALL NODES.
4070  DO 4080 J=1,NNODE
```

```
      IF (DELAMX.LT.DELAY(J)) DELMAX = DELAY(J)
4080  NODESW(J) = L1Y
C     SEND THE MESSAGES.
4090  RAREA(2) = SECNDS(0.) + DELMAX*3.
C     RECORD THE TIME OF, OR THE TIME SINCE INITIALIZATION.
      T1= RAREA(2) - T1
      IF (IAREA(1).EQ.6) GO TO 4091
      T1 = RAREA(2)
      NSERIE = NSERIE + 1
      IAREA(2) = NSERIE
4091  DO 4100  J=1,NNODE
      IF (NODESW(J).EQ.L1Y) CALL SNDNT (J+7,,8,IAREA)
4100  CONTINUE
      GO TO (2590,2820) LOC
C
C
C
C     COPY THE LATEST SERIES NUMBER TO THE EXPERIMENT DESCRIPTION
C     FILE, AND EXIT.
9999  WRITE (3'1) NEXPER, NSERIE, LENTBL
      CALL CLOSE (3)
      CALL EXIT
9018  TYPE 9019, IOSB1(1)
9019  FORMAT (' CLSNT FAILED ON CENCTL.  IOSB1(1)=',I3)
      GO TO 9999
      END
```

```
C       PROGRAM LOCCTL - LOCAL EXPERIMENT CONTROL ROUTINE.
C       WRITTEN BY R. STRAZDAS.
C
C       THIS PROGRAM COOPERATES WITH THE CENTRAL EXPERIMENT
C       CONTROL ROUTINE IN RUNNING THE EXPERIMENT.  ALL
C       COMMUNICATIONS BETWEEN THIS NODE AND THE CONTROL NODE GO
C       THROUGH THIS PROGRAM.  IT ACCEPTS THE EXPERIMENT CONTROL
C       TABLE, SYNCHRONIZES THE NODE CLOCK, SETS APPROPRIATE
C       ACCESS/CONTROL BITS, ACTIVATES GENERATORS AND TRACING,
C       AND COORDINATES DATA BASE RETRIEVAL.
C
C
        INTEGER UPDATE (13)
        REAL LENAVE(8), LENEXP(8)
C       VARIABLE X IS USED ONLY TO WORD ALIGH THE DATBLK COMMON
C       AREA.  GENST IS GENERATOR STATUS: 'R' WHEN ACTIVE,
C       'N' WHEN INACTIVE, 'D' WHEN BEING DEACTIVATED.
        BYTE MAIL(106),BAREA(200),GENST(8),MIRST(8),TIMBAS(8),
     X TGFUNC(8),TFUNC(8),TGNODE(6,8),ENABLD,L1D,L1N,X
        DIMENSION IOSB1(2),IAREA(100),MSTAT(3),RAREA(50),T2(10),
     X GEN(8)
        COMMON /GENBLK/IRAN1,IRAN2,LINKNO(8),GENST,TGNODE,LENAVE,
     X LENEXP,TIMAVE(8),TIMEXP(8),TIMBAS,CONAVE(8),CONEXP(8),
     X TGFUNC
     X /TGTBLK/LINKN1(8),MIRST,TFUNC
     X /TIMCOR/TIMCOR
     X /DATBLK/ENABLD,X,TIMER,NEXPER,NSERIE,
     X MSGLEN(8),SIGML(8),SIG2ML(8),
     X MSGTIM(8),SIGMT(8),SIG2MT(8),
     X NCON(8),SIGCM(8),SIG2CM(8),
     X NRNDEL(8),SIGRD(8),SIG2RD(8),RDMIN(8),RDMAX(8),
     X NDISCO(8),
     X NDELAY(8),SIGD(8),SIG2D(8),DELMIN(8),DELMAX(8),
     X NARTIM,SIGAT,SIG2AT,ATMIN,ATMAX
        EQUIVALENCE (MAIL(99),NEXPE),(MAIL(101),NSERI),(MAIL(103),
     X NUMGEN),(MAIL(105),NUMMIR),(BAREA,IAREA),
     X (RAREA,IAREA),(T2,IAREA)
        DATA L1D,L1R,L1N /'D','R','N'/
        DATA GEN/6RGEN001,6RGEN002,6RGEN003,6RGEN004,6RGEN005,
     X 6RGEN006,6RGEN007,6RGEN008/
        DATA DATUP,CPU /5RDATUP,3RCPU/
C
C       CONNECT TO CENTRAL CONTROL
C
        CALL ASNLUN (2,'NS',0,IDS)
        I = 0
        IF (IDS.NE.1) GO TO 908
        CALL OPNNTW (2,IOSB1,MSTAT,2)
        IF (IOSB1(1).NE.1) GO TO 900
        CALL GNDNTW (IOSB1,ITYPE,106,MAIL,,,)
        IF (IOSB1(1).NE.1) GO TO 902
        IF (ITYPE.NE.1) GO TO 914
```

```
      CALL ASNLUN (1,'NS',0,IDS)
      I = 1
      IF (IDS.NE.1) GO TO 908
      CALL ACCNTW (1,IOSB1,MAIL,,)
      IF (IOSB1(1).NE.1) GO TO 904
      TYPE 10, NEXPE, NSERIE, NUMGEN, NUMMIR
10    FORMAT (' LOCCTL: NEXPER=',I5,5X,'NSERIE=',I5,5X,
     X 'NUMGEN=',I3,5X,'NUMMIR=',I3)
C
C     RECEIVE GENERATOR INFORMATION
C
      IF (NUMGEN.EQ.0) GO TO 120
      DO 110 J=1,NUMGEN
      CALL RECNTW (1,IOSB1,34,IAREA)
      IF (IOSB1(1).NE.1) GO TO 910
C     EXTRACT PARAMETERS.
      LINKNO(J) = IAREA(1)
      GENST(J) = L1N
      DO 105 K=1,6
105   TGNODE(K,J) = BAREA(K+2)
      LENAVE(J) = RAREA(3)
      LENEXP(J) = RAREA(4)
      TIMAVE(J) = RAREA(5)
      TIMEXP(J) = RAREA(6)
      TIMBAS(J) = BAREA(33)
      CONAVE(J) = RAREA(7)
      CONEXP(J) = RAREA(8)
110   TGFUNC(J) = BAREA(34)
      IRAN1 = 0
      IRAN2 = 0
C
C     RECEIVE MIRROR INFORMATION
C
120   IF (NUMMIR.EQ.0) GO TO 200
      DO 130 J=1,NUMMIR
      CALL RECNTW (1,IOSB1,3,IAREA)
      IF (IOSB1(1).NE.1) GO TO 912
      LINKN1(J) = IAREA(1)
      MIRST(J) = L1N
130   TFUNC(J) = BAREA(3)
C
C     SYNCHRONIZE THE CLOCK WITH CENTRAL CONTROL.
C
200   TYPE 201
201   FORMAT (' LOCCTL: EXPERIMENT CONTROL TABLE RECEIVED.')
      DO 310 J=1,10
      CALL RECNTW (1,,100,IAREA)
      T2(J) = SECNDS(0.)
310   CALL SNDNTW (1,,100,IAREA)
C     RECEIVE THE CLOCK CORRECTION.
      CALL RECNTW (1,,4,IAREA)
      TIMCOR = RAREA(1)
```

```
      TYPE 311, TIMCOR
311   FORMAT (' LOCCTL: CORRECTED LOCAL TIME = LOCAL TIME ',
      X '+ ',F13.6,' SECONDS.')
C
C     INITIALIZE THE DATA BLOCK.
C
      IAREA(1) = 0
      NEXPER = NEXPE
      NSERIE = NSERI
      GO TO 1400
C
C     RECEIVE INSTRUCTIONS.
C
C     THE INSTRUCTION TYPE IS RECEIVED IN IAREA(1).
C     STATUS IS RETURNED IN IAREA(1).
C     A MESSAGE IS TYPED ON THE LOCAL TERMINAL AFTER EACH
C     COMMAND COMPLETES.
500   CALL RECNTW (1,IOSB1,20,IAREA)
      IF  (IOSB1(1).NE.1) GO TO 916
C
C     DECODE THE INSTRUCTION.
C
      GO TO (1100,1200,1300,1400,1500,1600,1700) IAREA(1)
C
C     ACTIVATE COMMAND
C
1100  IAREA(2) = 1
      DO 1120 J=1,NUMGEN
      IF (IAREA(J+2).EQ.0) GO TO 1120
C     ACTIVATE GENERATOR J, UNLESS IT IS ALREADY ACTIVE.
      IF GENST(J).NE.L1R) CALL REQUES (GEN(J),,IDS)
      IF (IDS.NE.1) GO TO 1110
      IAREA(J+2) = LINKNO(J)
      GENST(J) = L1R
      GO TO 1120
C     REQUEST FAILED.  RETURN THE LINK NUMBER TO CENCTL
C     AND TYPE STATUS.
1110  IAREA(J+2) = 0 - LINKNO(J)
      IAREA(2) = IAREA(2) + 1
      TYPE 1115, LINKNO(J), IDS
1115  FORMAT (' LOCCTL: REQUEST TO ACTIVATE LINK ',I2,
     X ' FAILED.  IDS=',I3)
1120  CONTINUE
      CALL SNDNT (1,IOSB1,20,IAREA)
      TYPE 1130, IAREA(2)-1, IAREA(J), J=3,NUMGEN+2)
1130  FORMAT (' LOCCTL: ',I3,' FAILURES ON ACTIVATING LINKS ',
     X 8(I3,1X))
1140  CALL WAITNT (,IOSB1)
      IF (IOSB1(1).NE.1) GO TO 918
      GO TO 500
C
C     DEACTIVATE COMMAND
```

```
C
1200  IAREA(2) = 1
C     TURN OFF STATUS FOR THE GENERATORS.  THEY WILL SHUT DOWN
C     WHEN THEY NOTICE THIS, AND RESET A GLOBAL EVENT FLAG TO
C     SHOW THIS.
      DO 1210 J=1,NUMGEN
      IF (IAREA(J+2).EQ.0.OR.GENST(J).EQ.L1N) GO TO 1210
      CALL CLREF (J+40)
      GENST(J) = L1D
1210  CONTINUE
      DO 1230 J=1,NUMGEN
      IF (GENST(J).NE.L1D) GO TO 1230
      CALL WAITFR (J+40)
      GENST(J) = L1N
      IAREA(J+2) = LINKNO(J)
1230  CONTINUE
      CALL SNDNT (1,IOSB1,20,IAREA)
      TYPE 1250, (IAREA(J), J=3,NUMGEN+2)
1250  FORMAT (' LOCCTL: DEACTIVATED LINKS ',8(I3,1X))
      GO TO 1140
C
C     EXIT COMMAND
C
C     DEACTIVATE ALL LINKS AND EXIT.
1300  DO 1310, J=1,NUMGEN
      IF (GENST(J).EQ.L1N) GO TO 1310
      CALL CLREF (J+40)
      GENST(J) = L1D
1310  CONTINUE
      DO 1320 J=1,NUMGEN
      IF (GENST(J).NE.L1D) GO TO 1320
      CALL WAITFR (J+40)
1320  CONTINUE
C     TELL DATUP TO QUIT.
      UPDATE(1) = 6
      CALL SEND (DATUP,UPDATE,,IDS)
      IF (IDS.NE.1) GO TO 906
1324  CALL RESUME (DATUP,IDS)
      IF (IDS.NE.1) GO TO 1326
      IF (IDS.NE.-8) GO TO 924
C     DATUP HAD NOT SUSPENDED ITSELF; IT IS HANDLING
C     ANOTHER REQUEST.
      CALL WFSNE
      GO TO 1324
C     TELL CPU TO QUIT.
1326  X = L1D
      IAREA(1) = 1
      CALL SNDNT (1,,2,IAREA)
      TYPE 1330
1330  FORMAT (' LOCCTL: EXITING.')
      CALL CLSNTW (IOSB1)
      IF (IOSB1(1).NE.1) GO TO 922
```

```
      GO TO 1000
C
C     INITITALIZE COMMAND
C
1400  ENABLD = L1D
      TIMER = 0.
      DO 1410 I=1,8
      MSGLEN(I) = 0
      SIGML(I) = 0.
      SIG2ML(I) = 0.
      MSGTIM(I) = 0
      SIGMT(I) = 0.
      SIG2MT(I) = 0.
      NCON(I) = 0
      SIGCM(I) = 0.
      SIG2CM(I) = 0.
      NRNDEL(I) = 0
      SIGRD(I) = 0.
      SIG2RD(I) = 0.
      NDISCO(I) = 0
      NDELAY(I) = 0
      SIGD(I) = 0.
      SIG2D(I) = 0.
      RDMIN(I) = 1.E6
      RDMAX(I) = 0.
      DELMIN(I) = 1.E6
1410  DELMAX(I) = 0.
      NARTIM = 0
      SIGAT = 0.
      SIG2AT = 0.
      ATMIN = 1.E6
      ATMAX = 0.
C     CHECK IF THIS WAS THE INITIAL INITIALIZE.
      IF (IAREA(1).NE.0) GO TO 1415
C     REQUEST 'CPU', THE EXPERIMENT TIMER TASK.
      X = L1R
      CALL REQUES (CPU,,IDS)
      IF (IDS.EQ.1) GO TO 1412
C     'CPU' WAS NOT INSTALLED.  INFORM CENCTL.
      TYPE 1411, IDS
1411  FORMAT (' LOCCTL: REQUEST TO RUN TASK ''CPU'' FAILED.',
     X '  IDS=',I3)
      IAREA(1) = 1
1412  IAREA(2) = 0
C     REQUEST DATUP, THE DATA AREA UPDATER.
      CALL REQUES (DATUP,,IDS)
      IF (IDS.EQ.1) GO TO 1414
C    DATUP WAS NOT INSTALLED.  INFORM CENCTL.
      TYPE 1413, IDS
1413  FORMAT (' LOCCTL: REQUEST TO RUN DATUP FAILED.',
     X '  IDS=',I3)
      IAREA(2) = 1
```

```
1414   CALL SNDNTW (1,,4,IAREA)
       GO TO 500
1415   NSERIE = IAREA(2)
1420   NTICKS = IFIX ((RAREA(2)-SECNDS(0.)-TIMCOR)*60.)
       IF (NTICKS.LT.0) GO TO 1430
       CALL MARK (9,NTICKS,1,IDS)
       IF (IDS.NE.1) GO TO 1440
       CALL WAITFR(9)
C      DATA AREA READY FOR UPDATING.
1430   ENABLD = L1R
       IAREA(1) = 1
       IF (NTICKS.LT.0) IAREA(1) = 1-NTICKS
       GO TO 1460
C      CHECK FOR INSUFFICIENT DYNAMIC STORAGE.
1440   IF (IDS.NE.-1) GO TO 1450
       CALL WFSNE
       GO TO 1420
1450   ENABLD = L1R
       IAREA(1) = 0
1460   CALL SNDNT (1,IDSB1,2,IAREA)
       IF (IAREA(1).EQ.0) GO TO 920
       TYPE 1470
1470   FORMAT (' LOCCTL: INITIALIZED.')
       IF (NTICKS.LT.0) TYPE 1480, 0-NTICKS
1480   FORMAT (' LOCCTL: BUT LATE BY ',I5,'/60 SECONDS.')
       GO TO 1140
C
C      NOTRACE COMMAND
C
1500   GO TO 500
C
C      RETRIEVE COMMAND
C
1600   IAREA(2) = NSERIE
       NTICKS = IFIX((RAREA(2)-SECNDS(0.)-TIMCOR)*60.)
       IF (NTICKS.LT.0) GO TO 1605
       CALL MARK (9,NTICKS,1,IDS)
       IF (IDS.NE.1) GO TO 1607
       CALL WAITFR(9)
       IAREA(1) = 1
       GO TO 1610
1605   IAREA(1) = 1-NTICKS
       GO TO 1610
1607   IAREA(1) = 0
C      DISABLE UPDATING AND SEND DATA TO CENTRAL CONTROL.
1610   ENABLD = L1D
       DO 1620 I=1,8
       IAREA(3) = MSGLEN(I)
       IAREA(4) = MSGTIM(I)
       IAREA(5) = NCON(I)
       IAREA(6) = NRNDEL(I)
       IAREA(7) = NDISCO(I)
```

```
      IAREA(8) = NDELAY(I)
      RAREA(5) = SIGML(I)
      RAREA(6) = SIG2ML(I)
      RAREA(7) = SIGMT(I)
      RAREA(8) = SIG2MT(I)
      RAREA(9) = SIGCM(I)
      RAREA(10) = SIG2CM(I)
      RAREA(11) = SIGRD(I)
      RAREA(12) = SIG2RD(I)
      RAREA(13) = RDMIN(I)
      RAREA(14) = RDMAX(I)
      RAREA(15) = SIGD(I)
      RAREA(16) = SIG2D(I)
      RAREA(17) = DELMIN(I)
      RAREA(18) = DELMAX(I)
1620  CALL SNDNTW (1,,72,IAREA)
      IAREA(3) = NARTIM
      RAREA(3) = SIGAT
      RAREA(4) = SIG2AT
      RAREA(5) = ATMIN
      RAREA(6) = ATMAX
      RAREA(7) = TIMER
      CALL SNDNTW (1,IOSB1,72,IAREA)
      IF (IOSB1(1).NE.1) GO TO 918
      TYPE 1630
1630  FORMAT (' LOCCTL: DATA AREA RETRIEVED.')
      IF (NTICKS.LT.0) TYPE 1480, 0-NTICKS
      GO TO 500
C
C     TRACE COMMAND
C
1700  GO TO 500
C
C
C
1000  CALL EXIT
900   TYPE 901, IOSB1(1)
901   FORMAT (' OPNNT UNSUCCESSFUL ON LOCCTL.  IOSB1(1)= ',I3)
      GO TO 1000
902   TYPE 903, IOSB1
903   FORMAT (' GNDNTW UNSUCCESSFUL ON LOCCTL.  IOSB1= ',
     X I3,', ',I3)
      GO TO 1000
904   TYPE 905, I, IOSB1(1)
905   FORMAT (' ACCNTW(',I2,') UNSUCCESSFUL ON LOCCTL.',
     X '  IOSB1(1)= ',I3)
      GO TO 1000
906   TYPE 907, IDS
907   FORMAT (' SEND TO DATUP FAILED ON LOCCTL.  IDS=',I3)
      GO TO 1000
908   TYPE 909, I, IDS
909   FORMAT (' ASNLUN FAILED ON LOCCTL.  I=',I2,5X,'IDS=',I3)
```

```
      GO TO 1000
910   TYPE 911, J, IOSB1(1)
911   FORMAT (1X,I2,'TH GEN RECW FAILED ON LOCCTL.  IOSB1(1)=',I3)
      GO TO 1000
912   TYPE 913, IOSB1(1)
913   FORMAT (' MIR RECW FAILED ON LOCCTL.  IOSB1(1)=',I3)
      GO TO 1000
914   TYPE 915, ITYPE
915   FORMAT (' LOCCTL GNDNTW:  ITYPE=',I3)
      GO TO 1000
916   TYPE 917, IOSB1(1)
917   FORMAT (' RECW INSTRUCTION ON LOCCTL FAILED.  IOSB1(1)=',I3)
      GO TO 1000
918   TYPE 919, IOSB1(1)
919   FORMAT (' SND STATUS ON LOCCTL FAILED.  IOSB1(1)=',I3)
      GO TO 1000
920   TYPE 921, IDS
921   FORMAT (' CALL MARK FAILED ON LOCCTL.  IDS=',I3)
      GO TO 1000
922   TYPE 923, IOSB1(1)
923   FORMAT (' CLSNTW FAILED ON LOCCTL.  IOSB1(1)=',I3)
      GO TO 1000
924   TYPE 925, IDS
925   FORMAT (' RESUME DATUP ON LOCCTL FAILED.  IDS=',I3)
      GO TO 1000
      END
```

```
C       PROGRAM GEN1
C       WRITTEN BY R. STRAZDAS.
C
C       THIS PROGRAM READS TRAFFIC GENERATOR PARAMETERS FROM A
C       COMMON DATA AREA AND USES THEM TO DECIDE HOW MUCH TRAFFIC,
C       WHEN, WHERE, ETC. TO SEND THROUGH THE NETWORK.  IT
C       REQUIRES A MIRROR TASK AT THE OTHER END OF THE NETWORK
C       LINK.  DATA COLLECTED IS STORED IN A COMMON AREA.
C
C
        LOGICAL*1 LOG
        INTEGER UPDATE(13)
        BYTE TGTBLK(72),NODE(6),NAME(6),TSKNAM(6),TSKNM6,OUTMSG(4)
        LOGICAL*1 L1N,L1R,L1W,L1D,RBUFF(1000)
        REAL LENAVE(8), LENEXP(8), RAREA(250), RUPDAT(4)
        BYTE GENST(8), TIMBAS(8), TGFUNC(8), TFUNC, TGNODE(6,8)
        DIMENSION IOSB1(2), IOSB2(2), IAREA(500),
       X GETBUF(8), MSTAT(3)
        COMMON /GENBLK/IRON1,IRON2,LINKNO(8),GENST,TGNODE,LENAVE,
       X LENEXP,TIMAVE(8),TIMEXP(8),TIMBAS,CONAVE(8),
       X CONEXP(8),TGFUNC
       X /TIMCOR/TIMCOR
       X /RECEIV/ME,NRECNT,RBUFF
        EQUIVALENCE (TSKNM6,(TSKNAM(6)), (OUTMSG(1),LINKN),
       X (OUTMSG(3),NMSG), (RAREA,IAREA), (RUPDAT,UPDATE(6))
        DATA NAME/'M','I','R','R','O','R'/
        DATA L1N,L1R,L1W,L1D/'N','R','W','D'/
        DATA DATUP /5RDATUP/
C
C       GET THE TASK NAME SO THIS GENERATOR WILL KNOW WHICH
C       PART OF THE DATA AREA IT SHOULD ACCESS.
C
        CALL GETTSK (GETBUF)
        CALL R50ASC (6,GETBUF,TSKNAM)
        DECODE (1,2,TSKNM6,ERR=920) ME
2       FORMAT (I1)
        UPDATE(2) = ME
        CALL ASNLUN (2,'NS',0,IDS)
        IF (IDS.NE.1) GO TO 908
        CALL OPNNTW (2,IOSB1,MSTAT,1)
        IF (IOSB1(1).NE.1) GO TO 900
        CALL ASNLUN (1,'NS',0,IDS)
        IF (IDS.NE.1) GO TO 908
C
C       SET UP THE TARGET BLOCK FOR THE CONNECT CALL.
C
50      CALL BACC (LOG,TGTBLK,,,,,,)
        IF (LOG.EQ..FALSE.) GO TO 914
        DO 10 I=1,6
10      NODE(I) = TGNODE(I,ME)
        CALL BFMT1 (LOG,TGTBLK,6,NODE,,6,NAME)
        IF (LOG.EQ..FALSE.) GO TO 916
```

```
C       INITIALIZE THE NUMBER OF PENDING RECEIVES.
        NRECNT = 0
C       IF TIMING DEPENDS ON WAITING FOR REPLIES FROM THE
C       MIRROR TASK, THEN USE EVENT FLAG 10 TO COORDINATE IT.
        IF (TIMBAS(ME).EQ.L1R) CALL SETEF (10)
C       INITIALIZE EVENT FLAG 9.
        IF (TIMBAS(ME).EQ.L1W) CALL SETEF (9)
C       GET THE NUMBER OF MESSAGES PER CONNECT.
        NMSG = IFIX (EXPON (CONAVE(ME),CONEXP(ME) + .99999)
        LINKN = LINKNO(ME)
C       NMSG AND LINKN ARE INCLUDED IN THE CONNECT MESSAGE.
15      CALL CONNTW (1,IOSB1,TGTBLK,4,OUTMSG,,)
        IF (IOSB1(1).NE.-7.OR.IOSB1(1).NE.33) GO TO 19
        TYPE 7888
7888    FORMAT (' RETRY CONNECT...')
        GO TO 15
19      IF (IOSB1(1).NE.1) GO TO 902
C       RECORD INFO.
        UPDATE(1) = 1
        UPDATE(3) = NMSG
        CALL SEND (DATUP,UPDATE,,IDS)
        IF (IDS.NE.1) GO TO 906
20      CALL RESUME (DATUP,IDS)
        IF (IDS.EQ.1) GO TO 40
        IF (IDS.NE.-8) GO TO 922
C       DATUP HAS NOT SUSPENDED ITSELF; IT IS HANDLING
C       ANOTHER REQUEST.
        CALL WFSNE
        GO TO 20
40      T = SECNDS(0.)
C
C       GENERATE AND SEND THE MESSAGES.
C
        DO 500 N=1,NMSG
C       CHECK FOR DEACTIVATE.
        IF (GENST(ME).EQ.L1D) GO TO 550
C       GET THE MESSAGE LENGTH.
        MSGLEN = IFIX (EXPON (LENAVE(ME),LENEXP(ME)) + .5)
        IF (MSGLEN.LT.6) MSGLEN = 6
        IF (MSGLEN.GT.1000) MSGLEN = 1000
C       CONSTRUCT THE MESSAGE.
        RAREA(1) = SECNDS(0.) + TIMCOR
C
C
C       BRANCH DEPENDING ON THE VALUE OF TIMBAS.
        IF (TIMBAS(ME).EQ.L1W) GO TO 100
        IF (TIMBAS(ME).EQ.L1R) GO TO 200
C
C       TIMBAS=N (NO WAITING.  SEND IMMEDIATELY.)
C
        CALL SNDNT (1,,MSGLEN,IAREA)
        T = SECNDS (T)
```

```
      ITIME = IFIX (T + .5)
      GO TO 300
C
C     TIMBAS=W (WAIT FOR ACKNOWLEDGE OF PREVIOUS MESSAGE)
C
C     WAIT UNTIL THE MESSAGE SHOULD BE SENT.
100   CALL WAITFR (9)
      CALL SNDNTW (1,IOSB1,MSGLEN,IAREA)
      IF (IOSB1(1).NE.1) GO TO 904
C     GET THE INTERMESSAGE TIME.
      ITIME = MAX0(IFIX(EXPON(TIMAVE(ME),TIMEXP(ME))+.5),0)
      CALL MARK (9,ITIME,1)
      GO TO 300
C
C     TIMBAS=R (WAIT FOR REPLY OF PREVIOUS MESSAGE)
C
C     GET THE INTERMESSAGE TIME.
200   ITIME = MAX0(IFIX(EXPON(TIMAVE(ME),TIMEXP(ME))+.5),0)
C     FIRST WAIT FOR THE REPLY TO THE PREVIOUS MESSAGE.
      CALL WAITFR (10)
C     NOW WAIT THE REQUIRED AMOUNT OF TIME.
      CALL MARK (9,ITIME,1)
      CALL CLREF (10)
      CALL WAITFR (9)
      CALL SNDNT (1,,MSGLEN,IAREA)
300   NRECNT = NRECNT + 1
C     IF WAITING FOR REPLIES FROM THE MIRROR TASK, MAKE SURE
C     WE DON'T MISS THE LAST ONE.
      IF (N.EQ.NMSG.AND.TGFUNC(ME).EQ.L1R) CALL CLREF (11)
C     RECORD INFO.
      UPDATE(1) = 2
      UPDATE(3) = ITIME
      UPDATE(4) = MSGLEN
      CALL SEND (DATUP,UPDATE,,IDS)
      IF (IDS.NE.1) GO TO 906
310   CALL RESUME (DATUP,IDS)
      IF (IDS.EQ.1) GO TO 330
      IF (IDS.NE.-8) GO TO 922
      CALL WFSNE
      GO TO 310
330   IF (TGFUNC(ME).EQ.L1R) CALL RMAC
500   CONTINUE
C
C     IF THE MIRROR TASK REPLIUES, THEN WAIT FOR THE LAST
C     RECEIVE.
      IF (TGFUNC(ME).EQ.L1R) CALL WAITFR (11)
C     DISCONNECT AND RECORD INFO.
550   CALL DSCNTW (1,IOSB1,4,OUTMSG)
      IF (IOSB1(1).NE.1) GO TO 912
      IF (GENST(ME).EQ.L1D) GO TO 600
      UPDATE(1) = 4
      CALL SEND (DATUP,UPDATE,,IDS)
```

```
      IF (IDS.NE.1) GO TO 906
560   CALL RESUME (DATUP,IDS)
      IF (IDS.EQ.1) GO TO 50
      IF (IDS.NE.-8) GO TO 922
      CALL WFSNE
      GO TO 560
C     TELL LOCCTL THAT THE LINK IS DISCONNECTED, AND EXIT.
600   CALL SETEF (ME+40)
      CALL CLSNTW (IOSB1)
      IF (IOSB1(1).NE.1) GO TO 918
C
C
C
1000  CALL EXIT
900   TYPE 901, TSKNAM, IOSB1(1)
901   FORMAT (' OPNNT UNSUCCESSFUL ON ',6A1,'. IOSB1(1)=',I3)
      GO TO 1000
902   TYPE 903, TSKNAM, IOSB1
903   FORMAT (' CONNTW UNSUCCESSFUL ON ',6A1,'.  IOSB1= ',
     X I3,',',I3)
      GO TO 1000
904   TYPE 905, TSKNAM, IOSB1(1), TIMBAS(ME)
905   FORMAT (' SNDNT UNSUCCESSFUL ON ',6A1,'.  IOSB1(1)=',
     X I3,'.  TIMBAS=',A1)
      GO TO 1000
906   TYPE 907, TSKNAM, IDS
907   FORMAT (' SEND TO DATUP FAILED ON ',6A1,'  IDS=',I3)
      GO TO 1000
908   TYPE 909, TSKNAM, IDS
909   FORMAT (' ASNLUN FAILED ON ',6A1,'.   IDS=',I3)
      GO TO 1000
912   TYPE 913, TSKNAM, IOSB1(1)
913   FORMAT (' DSCNTW UNSUCCESSFUL ON ',6A1,'.  IOSB1(1)=',I3)
      GO TO 1000
914   TYPE 915, TSKNAM
915   FORMAT (' BACC FAILED ON ',6A1,'.')
      GO TO 1000
916   TYPE 917, TSKNAM
917   FORMAT (' BFMT1 FAILED ON ',6A1,'.')
      GO TO 1000
918   TYPE 919, TSKNAM, IOSB1(1)
919   FORMAT (' CLSNTW UNSUCCESSFUL ON ',6A1,'.  IOSB1(1)=',I3)
      GO TO 1000
920   TYPE 921, TSKNAM
921   FORMAT (' ''ME'' DECODE FAILED ON ',6A1,'.')
      GO TO 1000
922   TYPE 923, TSKNAM, IDS
923   FORMAT (' RESUME DATUP FAILED ON ',6A1,'.  IDS=',I3)
      GO TO 1000
      END
```

```
      FUNCTION EXPON (AVE,EXP)
C
C     THIS FUNCTION COMPUTES AN EXPONENTIALLY DISTRIBUTED
C     RANDOM VARIABLE WITH AVERAGE VALUE 'AVE' AND EXPONENTIAL
C     PARAMETER 'EXP'.  I.E. F(X) ='EXP'*(E**(-'EXP'*X))
C
C
      COMMON /GENBLK/ IRAN1,IRAN2
C     IF 'EXP' IS ZERO, A CONSTANT VALUE WILL BE RETURNED.
      IF (EXP.EQ.0) GO TO 900
C     GET AN EXPONENTIALLY DISTRIBUTED RANDOM NUMBER.
      T = RAN (IRAN1,IRAN1)
      T = ALOG (1./T)/EXP
C     ADJUST T SO THAT IT HAS THE PROPER AVERAGE VALUE.
      EXPON = T + AVE - 1./EXP
      RETURN
900   EXPON = AVE
      RETURN
      END
```

```
        ;RMAC DOES A NETWORK RECEIVE, SETTING UP 'RAST', AN AST
        ;WHICH IS INVOKED AT THE COMPLETION OF THE RECEIVE.
        ;
        .MCALL  REC$E,REC$,ASTX$S,QIOW$C,EXIT$S
        .PSECT  RECEIV,GBL,OVR,D
ME:     .BLKW   1
NRECNT: .BLKW   1
RBUFF:  .BLKB   1000.
        .PSECT
RMAC::  REC$E   RECBLK
        RETURN
STATUS: .BLKW   2
RECBLK: REC$    1,,STATUS,RAST,<RBUFF,1000.>
RAST:   CMPB    #IS.SUC,STATUS  ;WAS IT SUCCESSFUL?
        BNE     10$
        CALL    RFTN
        CMP     (SP)+,SP        ;CLEAN THE STACK
        ASTX$S
10$:    MOV     #STATUS,R2
        MOV     #IOBUF,R1
        MOV     #IOBUF,R0
        CALL    $EDMSG
        QIOW$C  IO.WLB,5,20,,,,<IOBUF,R1,40>
        ASTX$S
IOBUF:  .ASCIZ  /ERROR IN REC$E IN RMAC: %P/
        .END
```

```
      SUBROUTINE RFTN
C     SUBPROGRAM CALLED BY RMAC TO COORDINATE INFORMATION
C     FLOW AND TIMING AFTER A GENERATOR RECEIVES A MESSAGE.
      INTEGER UPDATE(13)
      DIMENSION RUPDAT(4)
      COMMON /RECEIV/ME,NRECNT,RBUFF(250)
     X /TIMCOR/TIMCOR
      EQUIVALENCE (RUPDAT,UPDATE(6))
      DATA DATUP /5RDATUP/
C
C     RECORD INFO.
C
      T = SECNDS(0.) + TIMCOR
      UPDATE(1) = 3
      UPDATE(2) = ME
      RUPDAT(1) = T - RBUFF(1)
      CALL SEND (DATUP,UPDATE,,IDS)
      IF (IDS.NE.1) GO TO 98
10    CALL RESUME (DATUP,IDS)
      IF (IDS.EQ.1) GO TO 20
      IF (IDS.NE.-8) GO TO 96
```

```
C       DATUP HAS NOT SUSPENDED ITSELF; IT IS HANDLING
C       ANOTHER REQUEST.
        CALL WFSNE
        GO TO 10
20      NRECNT = NRECNT - 1
        IF (NRECNT.LT.0) TYPE 100, ME, NRECNT
C       EVENT FLAG 10 INFORMS THE GENERATOR THAT THE MESSAGE
C       HAS BEEN RECEIVED.
        CALL SETEF (10)
C       EVENT FLAG 11 INFORMS THE GENERATOR THAT THE LAST
C       MESSAGE HAS BEEN RECEIVED.
        IF (NRECNT.EQ.0) CALL SETEF (11)
        RETURN
96      TYPE 97, IDS
97      FORMAT (' RESUME DATUP FAILED ON RFTN.  IDS=',I3)
        RETURN
98      TYPE 99, IDS
99      FORMAT (' SEND TO DATUP FAILED ON RFTN.  IDS=',I3)
        RETURN
100     FORMAT (' RFTN: ',I2,'TH RECEIVE COUNT IS ',I3)
        END
```

```
C       PROGRAM MIRROR
C       WRITTEN BY R. STRAZDAS.
C
C       THIS PROGRAM COMPLEMENTS THE TRAFFIC GENERATOR BY
C       PROVIDING A TARGET TASK WHICH CAN RETURN THE DATA IT
C       RECEIVES.  NO MORE THAN ONE OF THIS TASK WILL RESIDE
C       IN EACH NODE, AND IT WILL ACCEPT ALL CONNECT REQUESTS
C       FROM GENERATORS.
C
C
        INTEGER UPDATE(13)
        BYTE MAIL(102),L1R,L1N
        BYTE TGFUNC(8),TGSTAT(8)
        DIMENSION IAREA(500),MSTAT(3),NMSG(8),RAREA(250),
       X RUPDAT(4),IOSB1(2),IOSB2(2),IOSB3(2),IOSB4(2),IOSB5(2),
       X IOSB6(2), IOSB7(2), IOSB8(2), IOSB9(2), IOSB(2,9)
        COMMON /TGTBLK/LINKNO(8),TGSTAT,TGFUNC
       X /TIMCOR/TIMCOR
        EQUIVALENCE (LINKC,MAIL(99)), (NMESS,MAIL(101)),
       X (LINKD,MAIL(1)), (RAREA,IAREA)
        EQUIVALENCE (IOSB1,IOSB(1,1)), (IOSB2,IOSB(1,2)),
       X (IOSB3,IOSB(1,3)), (IOSB4,IOSB(1,4)), (IOSB5,IOSB(1,5)),
       X (IOSB6,IOSB(1,6)), (IOSB7,IOSB(1,7)), (IOSB8,IOSB(1,8)),
       X (IOSB9,IOSB(1,9)), (RUPDAT,UPDATE(6))
        DATA L1R,L1N/'R','N'/
        DATA DATUP /5RDATUP/
        CALL ASNLUN (2,'NS',0,IDS)
        IF (IDS.NE.1) GO TO 910
C       DETERMINE HOW MANY LINKS THERE MAY BE.
        DO 50 I=1,10
        IF (LINKNO(I).EQ.0) GO TO 60
50      CONTINUE
60      CALL OPNNTW (2,IOSB1,MSTAT,I-1)
        IF (IOSB1(1).NE.1) GO TO 900
        UPDATE(1) = 5
        NLINKS = 0
        T2 = SECNDS(0.)
C       INITIALIZE IOSB1 SINCE IT WAS USED WITH OPNNT.
        IOSB1(1) = 0
100     CALL GNDNT (IOSB9,ITYPE,102,MAIL,,,)
C
C       WAIT FOR SOMETHING TO HAPPEN.
C
200     CALL WAITNT (INDEX,IOSB1,IOSB2,IOSB3,IOSB4,IOSB5,
       X IOSB6,IOSB7,IOSB8,IOSB9)
        T = SECNDS(0.) + TIMCOR
        T1 = SECNDS(T2)
        T2 = SECNDS(0.)
        IF (INDEX.EQ.9) GO TO 500
C
C       A RECEIVE HAS COMPLETED.
C
```

```
      IF (IOSB(1,INDEX).EQ.1) GO TO 250
C     THE RECEIVE WAS UNSUCCESSFUL.  EITHER A DISCONNECT WAS
C     RECEIVED OF A PROBLEM EXISTS.
      IF (IOSB(1,INDEX).NE.-3) GO TO 904
      IF (IOSB9(1).EQ.0) GO TO 912
      GO TO 500
C     RETURN THE MESSAGE?
250   IF (TGFUNC(INDEX).NE.L1R) GO TO 300
      CALL SNDNT (INDEX+7,,IOSB(2,INDEX),IAREA)
300   NMSG(INDEX) = NMSG(INDEX) - 1
C
C     RECORD INFO
C
      UPDATE(2) = INDEX
      RUPDAT(1) = T - RAREA(1)
      RUPDAT(2) = T1
      CALL SEND (DATUP,UPDATE,,IDS)
      IF (IDS.NE.1) GO TO 908
301   CALL RESUME (DATUP,IDS)
      IF (IDS.EQ.1) GO TO 305
      IF (IDS.NE.-8) GO TO 916
C     DATUP HAS NOT SUSPENDED ITSELF; IT IS HANDLING
C     ANOTHER REQUEST.
      CALL WFSNE
      GO TO 301
305   IOSB(1,INDEX) = 0
      IF (NMSG(INDEX).EQ.0) GO TO 200
      GO TO (310,320,330,340,350,360,370,380), INDEX
310   CALL RECNT (INDEX+7,IOSB1,1000,IAREA)
      GO TO 200
320   CALL RECNT (INDEX+7,IOSB2,1000,IAREA)
      GO TO 200
330   CALL RECNT (INDEX+7,IOSB3,1000,IAREA)
      GO TO 200
340   CALL RECNT (INDEX+7,IOSB4,1000,IAREA)
      GO TO 200
350   CALL RECNT (INDEX+7,IOSB5,1000,IAREA)
      GO TO 200
360   CALL RECNT (INDEX+7,IOSB6,1000,IAREA)
      GO TO 200
370   CALL RECNT (INDEX+7,IOSB7,1000,IAREA)
      GO TO 200
380   CALL RECNT (INDEX+7,IOSB8,1000,IAREA)
      GO TO 200
C
C     A GET NETWORK DATA COMPLETED.
C
500   IF (IOSB9(1).NE.1) GO TO 902
C     CHECK THE TYPE OF DATA.
      IF (ITYPE.EQ.1) GO TO 600
      IF (ITYPE.EQ.3) GO TO 700
C     UNEXPECTED DATA TYPE
```

```
      TYPE 510, ITYPE, IOSB9
510   FORMAT (' UNEXPECTED GND IN MIRROR.  ITYPE=',I3,
     X ',  STATUS=',I3,',',I3)
      GO TO 100
C
C     CONNECT MESSAGE RECEIVED
C
600   INDEX = NLINK(LINKC)
      IF (INDEX.EQ.0) GO TO 650
      NMSG(INDEX) = NMESS
      CALL ASNLUN (INDEX+7,'NS',0,IDS)
      IF (IDS.NE.1) GO TO 910
      CALL ACCNTW (INDEX+7,IOSB9,MAIL,,)
      IF (IOSB9(1).NE.1) GO TO 914
C     STATUS IS 'READY'.
      TGSTAT(INDEX) = L1R
      NLINKS = NLINKS + 1
      CALL GNDNT (IOSB9,ITYPE,102,MAIL,,,)
C     ISSUE THE FIRST RECEIVE.
      GO TO 305
650   TYPE 660, (MAIL(J), J=25,44)
660   FORMAT (' UNAUTHORIZED CONNECT RECEIVED BY MIRROR FROM ',
     X 20A1)
      CALL REJNT (,MAIL,,)
      GO TO 100
C
C     DISCONNECT MESSAGE RECEIVED
C
700   INDEX = NLINK(LINKD)
      TGSTAT(INDEX) = L1N
      NLINKS = NLINKS - 1
      IF (NMSG(INDEX).NE.0) TYPE 760,LINKNO(INDEX),NMSG(INDEX)
760   FORMAT (' LINK',I3,' DISCONNECTED FROM MIRROR WITH ',
     X I3,' MESSAGES OUTSTANDING.')
      IF (NLINKS.NE.0) GO TO 100
C
C     EXIT.
C
      CALL CLSNTW (IOSB1)
      IF (IOSB1(1).NE.1) GO TO 906
1000  CALL EXIT
900   TYPE 901, IOSB1(1)
901   FORMAT (' OPNNT UNSUCCESSFUL ON MIRROR.  IOSB1(1)=',I3)
      GO TO 1000
902   TYPE 903, IOSB9
903   FORMAT (' GNDNT UNSUCCESSFUL ON MIRROR.  IOSB9=',
     X I3,', ',I3)
      GO TO 1000
904   TYPE 905, LINKNO(INDEX), IOSB(1,INDEX), NMSG(INDEX)
905   FORMAT (' RECNT ON LINK',I3,' UNSUCCESSFUL ON MIRROR.',
     X '  IOSB1(1)=',I3/5X,'OUTSTANDING MESSAGES =',I3)
      GO TO 1000
```

```
906   TYPE 907, IOSB1(1)
907   FORMAT (' CLSNTW UNSUCCESSFUL ON MIRROR.  IOSB1(1)=',I3)
      GO TO 1000
908   TYPE 909, IDS
909   FORMAT (' SEND TO DATUP FAILED ON MIRROR.  IDS=',I3)
      GO TO 1000
910   TYPE 911, IDS
911   FORMAT (' CALL ASNLUN FAILED ON MIRROR.  IDS=',I3)
      GO TO 1000
912   TYPE 913, IOSB9(1), NMSG(INDEX)
913   FORMAT (' RECNT FAILED, WITHOUT DISCONNECT, ON MIRROR.',
     X '  IOSB9=',I3/5X,'OUTSTANDING MESSAGES =',I3)
      GO TO 1000
914   TYPE 915, IOSB9(1)
915   FORMAT (' ACCNTW FAILED ON MIRROR.  IOSB9(1)=',I3)
      GO TO 1000
916   TYPE 917, IDS
917   FORMAT (' RESUME DATUP FAILED ON MIRROR.  IDS=',I3)
      GO TO 1000
      END



      FUNCTION NLINK (LINK)
C     THIS SUBPROGRAM RETURNS THE INDEX OF THE ARRAY LINKNO
C     WHERE A GIVEN LINK NUMBER IS FOUND.  IF IT IS NOT FOUND,
C     0 IS RETURNED.
      COMMON /TGTBLK/LINKNO(8)
      DO 100 I=1,8
      IF (LINKNO(I).EQ.LINK) GO TO 200
100   CONTINUE
      NLINK = 0
      RETURN
200   NLINK = I
      RETURN
      END
```

```
C       PROGRAM DATUP - DATA BASE UPDATER.
C       WRITTEN BY R. STRAZDAS.
C
C       THIS PROGRAM WILL RESIDE IN EACH NODE AND IS SOLELY
C       RESPONSIBLE FOR UPDATING THE DATA BASE.  THE LOCAL
C       EXPERIMENT CONTROL ROUTINE WILL CALL IT TO INITIALIZE
C       THE DATA BASE, AND IT ALWAYS CHECKS ITS ENABLE BIT BEFORE
C       WRITING.  IT IS ALSO USED BY THE GENERATOR(S), THE
C       MIRROR TASK, AND THE MONITOR TASK.
C
C
        BYTE ENABLD, L1D, X
        INTEGER UPDATE(15)
        DIMENSION RUPDAT(4)
C       VARIABLE X IS USED ONLY TO PRESERVE WORD ALIGNMENT.
        COMMON /DATBLK/ENABLD, X, TIMER, NEXPE, NSERI,
       X MSGLEN(8), SIGML(8), SIG2ML(8),
       X MSGTIM(8), SIGMT(8), SIG2MT(8),
       X NCON(8), SIGCM(8), SIG2CM(8),
       X NRNDEL(8), SIGRD(8), SIG2RD(8), RDMIN(8), RDMAX(8),
       X NDISCO(8),
       X NDELAY(8), SIGD(8), SIG2D(8), DELMIN(8), DELMAX(8),
       X NARTIM, SIGAT, SIG2AT, ATMIN, ATMAX
        EQUIVALENCE (RUPDAT,UPDATE(8))
        DATA L1D /'D'/
C       GET SOME DATA.
C
C       SUSPEND MYSELF, AND WAIT FOR SOMEONE TO SEND TO AND
C       RESUME ME.
50      CALL SUSPND
        CALL RECEIV (,UPDATE,,IDS)
        IF (IDS.EQ.1) GO TO 99
        IF (IDS.NE.-8) TYPE 60, IDS
60      FORMAT (' DATUP: CALL RECEIV FAILED.  IDS=',I3)
        GO TO 50
C       CHECK IF DATA UPDATE IS DISABLED.
99      IF (UPDATE(3).EQ.6) CALL EXIT
        IF (ENABLD.EQ.L1D) GO TO 50
        N = UPDATE(4)
        IF (UPDATE(3).LE.0.OR.UPDATE(3).GE.6) GO TO 998
        GO TO (100,200,300,400,500), UPDATE(3)
C       CONNECT MESSAGE SENT BY GENERATOR N.
100     NCON(N) = NCON(N) + 1
        SIGCM(N) = SIGCM(N) + FLOAT(UPDATE(5))
        SIG2CM(N) = SIG2CM(N) + FLOAT(UPDATE(5)**2)
        GO TO 50
C       MESSAGE SENT BY GENERATOR N.
200     T = FLOAT (UPDATE(5))/60.
        MSGTIM(N) = MSGTIM(N) + 1
        SIGMT(N) = SIGMT(N) + T
        SIG2MT(N) = SIG2MT(N) + T*T
        MSGLEN(N) = MSGLEN(N) + 1
```

```
      SIGML(N) = SIGML(N) + UPDATE(6)
      SIG2ML(N) = SIG2ML(N) + UPDATE(6)*UPDATE(6)
      GO TO 50
C     MESSAGE RECEIVED BY GENERATOR N.
300   NRNDEL(N) = NRNDEL(N) + 1
      SIGRD(N) = SIGRD(N) + RUPDAT(1)
      SIG2RD(N) = SIG2RD(N) + RUPDAT(1)*RUPDAT(1)
      IF (RDMIN(N).GT.RUPDAT(1)) RDMIN(N) = RUPDAT(1)
      IF (RDMAX(N).LT.RUPDAT(1)) RDMAX(N) = RUPDAT(1)
      GO TO 50
C     DISCONNECT SENT BY GENERATOR N.
400   NDISCO(N) = NDISCO(N) + 1
      GO TO 50
C     MESSAGE RECEIVED BY MIRROR ON ITS NTH LINK.
500   NDELAY(N) = NDELAY(N) + 1
      SIGD(N) = SIGD(N) + RUPDAT(1)
      SIG2D(N) = SIG2D(N) + RUPDAT(1)*RUPDAT(1)
      IF (DELMIN(N).GT.RUPDAT(1)) DELMIN(N) = RUPDAT(1)
      IF (DELMAX(N).LT.RUPDAT(1)) DELMAX(N) = RUPDAT(1)
      NARTIM = NARTIM + 1
      SIGAT = SIGAT + RUPDAT(2)
      SIG2AT = SIG2AT + RUPDAT(2)+RUPDAT(2)
      IF (ATMIN.GT.RUPDAT(2)) ATMIN = RUPDAT(2)
      IF (ATMAX.LT.RUPDAT(2)) ATMAX = RUPDAT(2)
      GO TO 50
998   TYPE 999, UPDATE(3)
999   FORMAT (' DATUP: INVALID INDEX: ',I2)
      GO TO 50
      END
```

```
C       PROGRAM CPU.
C       WRITTEN BY R. STRAZDAS.
C
C       THIS PROGRAM MONITORS CPU USAGE BY INCREMENTING A
C       COUNTER IN THE DATA BASE, PROPORTIONAL TO THE TIME
C       THIS LOW-PROIORITY PROGRAM RUNS.
C
C
        BYTE ENABLD, X, L1D
        COMMON /DATBLK/ ENABLD, X, TIMER
        DATA L1D /'D'/
10      DO 20, J=1,320
        K = J
20      CONTINUE
        IF (ENABLD.NE.L1D) TIMER = TIMER + 1
        IF (X.EQ.L1D) CALL EXIT
        GO TO 10
        END
```

```
C       PROGRAM KOMMON.
C       WRITTEN BY R. STRAZDAS.
C
C       THIS ROUTINE DEFINES A SYSTEM COMMON AREA FOR USE IN
C       THE NETWORK TRAFFIC GENERATOR PROGRAMS CENCTL, LOCCTL,
C       GEN1, MIRROR, AND DATUP.
C
C
        BLOCK DATA
        REAL LENAVE(8), LENEXP(8)
        BYTE GENST(8),MIRST(8), TIMBAS(8),TGFUNC(8),TFUNC(8),
       X TGNODE(6,8), ENABLD, X
C       VARIABLE X IS USED ONLY TO WORD ALIGN COMMON BLOCK
C       DATBLK.
        COMMON /GENBLK/IRAN1,IRAN2,LINKNO(8),GENST,TGNODE,LENAVE,
       X LENEXP,TIMAVE(8),TIMEXP(8),TIMBAS,CONAVE(8),
       X CONEXP(8), TGFUNC
       X /TGTBLK/ LINKN1(8), MIRST, TFUNC
       X /TIMCOR/ TIMCOR
       X /DATBLK/ ENABLD, X, TIMER, NEXPE, NSERI,
       X MSGLEN(8), SIGML(8), SIG2ML(8),
       X MSGTIM(8), SIGMT(8), SIG2MT(8),
       X NCON(8), SIGCM(8), SIG2CM(8),
       X NRNDEL(8), SIGRD(8), SIG2RD(8), RDMIN(8), RDMAX(8),
       X NDISCO(8),
       X NDELAY(8), SIGD(8), SIG2D(8), DELMIN(8), DELMAX(8),
       X NARTIM, SIGAT, SIG2AT, ATMIN, ATMAX
        END
```